Lecture-28
SSk

04/04/18

Transaction management

Recoverability

Cascadeless schedules

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read (A)	
	write(A)	
		read(A)

schedule 12

if this transaction fails & needs to be rolled back

then these two also need to be rolled back as they are sequential

] Cascading rollback → very costly, should be avoided

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
commit		
	read(A)	
	write(A)	
	commit	
		read (A)

can't rollback anymore

Cascadeless schedule → schedule not having sequencial transactions that can cause cascading rollback.

We can add 'commit' at the end of each transaction to make cascadeless schedule.

If T_i and T_j are sequencial transactions then read in T_j will occur after commit of T_i.

Scanned by CamScanner

Implementation of isolation

Implement locks on data items to make it isolated from other transactions.

Locks

1. Shared → S(Q) ⇒ read(Q)
2. Exclusive → X(Q) ⇒ read(Q), write(Q)

Concurrency control manager → manages lock on data items

Lock compatibility function -

T_i

T_j T_j → B(Q)

T_j has acquired lock on data item Q using lock mode B

T_i → A(Q)

T_i has acquired lock on Q using lock mode A

⇒ This means, lock mode A and B are compatible with each other.

	S	X
S	True	False
X	False	False

LCM - lock compatibility matrix

T_1	T_2
lock-X(B);	lock-S(A);
read(B);	read(A);
B: = B-50;	unlock(A);
write(B);	lock-S(B);
lock-X(A);	read(B);
read(A);	unlock(B);
A := A +50;	display(A+B);
write(A);	
unlock(A);	

Scanned by CamScanner

T₁	T₂	concurrency control manager
lock-X(B)		~~grant-X(B, T₁)~~
		grant-X(B, T₁)
read(B)		
B:=B-50		
write(B)		
unlock(B)		
	lock-S(A)	
		grant-S(A, T₂)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant ~~B~~-S(B, T₂)
	read(B)	
	unlock(B)	
	display(A+B)	
		grant-X(A, T₂)
lock-X(A)		
read(A)		
A:=A+50		
write(A)		
unlock(A)		