1: Serializability and 2PL

Consider the following schedule.

	T1	T2	T3
0	start		
1	read B		
2	write B		
3		start	
4		read B	
5		write B	
6			start
7			read A
8			write A
9		read A	
10		write A	
11		COMMIT	
12	read D		
13	write D		
14			COMMIT
15	COMMIT		

(a) What serial schedule is this equivalent to? If none, then explain why.

Solution: The serializability graph for the above schedule is: T1 → T2 ← T3. Any order that complies with the topological order of the graph, such as T1 → T3 → T2 or T3 → T1 → T2, is an equivalent serial schedule for our schedule.

(b) Explain whether this schedule is consistent with two phase locking? Transactions can get the lock on a data item anytime in the schedule before accessing the data item. If your answer is yes, insert into the schedule a minimal set of additional operations that will make the schedule no longer consistent with two phase locking. Do not introduce any new transaction. If your answer is no, then remove from the schedule a minimal set of operations, so that the revised schedule is consistent with two phase locking.

Solution: One solution for this problem is for T1 to get exclusive locks on B and D at the beginning of the transaction (i.e. immediately after start), and releases the lock on B and D immediately after it writes on B and D, respectively. Then, the schedule will be 2PL. If we add "T3 write A" after step 10 in the schedule, it will not be 2PL anymore. In the resulting schedule, T2 and T3 will not be able to acquire the locks they need.

(c) Consider the version of the schedule that is consistent with two phase locking (either the original one or your revised version, depending on your answer to the previous part). Is that version consistent with strict two phase locking? Why or why not?

Solution: It is not because transactions do not release their locks after they commit.

2: Locking & Degree of Consistency

Consider the following locking protocol: Before a transaction T writes a data object A, T acquires an exclusive lock on A, and holds onto this lock till the end of the transaction. Before a transaction T reads a data object A, T acquires a shared lock on A, but releases the lock immediately after reading A. Write a schedule where each transaction follows this locking protocol, but the schedule is not serializable. Explain the degree of consistency that each transaction observes in your example.

Solution: Consider the following schedule. Each transaction in the schedule follows the proposed locking protocol: T2 gets an exl. Its serialization graph contains edges T1 → T2 and T2 → T1, therefore it is not serializable. T2 observes consistency degree 3 as it puts long lock on each item it writes and does not read any item. T1 observes consistency degree 2 as it uses short locks on reads. We can also justify these consistency degrees based on the concept of dirty data as follows. T2 does not overwrite dirty data of other transactions, commits its writes after the end of transaction, does not read dirty data from other transaction, and other transactions do not dirty its data, so T1 observes consistency degree 3. As T2 does not have any write, it already sees consistency degree 2. But, its reads dirty data from T2, so it does **not** observed consistency degree 3.

	T1	T2
0	start	
1	S.lock (A)	
2	read A	
3	S.releaseLock(A)	
4		start
5		X.lock(A)
7		write A
6		COMMIT
7		X.releaseLock(A)
8	S.lock(A)	
9	read A	
10	S.releaseLock(A)	
11	COMMIT	

3: Multi-granularity locking (Problem 17.11 in Cow Book)

Consider a database organized in terms of the following hierarchy of objects: The database itself is an object (D), and it contains two files (F_1 and F_2), each of which contains 1000 pages ($P_1, \cdots,$ P_{1000} and $P_{1001}, \cdots, P_{2000}$, respectively). Each page contains 100 records, and records are identified as p : i, where p is the page identifier and i is the slot of the record on that page. Multiple-granularity locking is used, with S, X, IS, IX and SIX locks, and database level, file-level, page-level and record-level locking. For each of the following operations, indicate the sequence of lock requests that must be generated by a transaction that wants to carry out (just) these operations:

1. Read record P_{1200} : 5.

2. Read records P_{1200} : 98 through P_{1205} : 2.
3. Read all (records on all) pages in file F_1.
4. Read pages P_{500} through P_{520}.
5. Read pages P_{10} through P_{980}.
6. Read all pages in F_1 and (based on the values read) modify 10 pages in F_1.
7. Delete record P_{1200}:98. (This is a write.)
8. Delete the first record from each page. (Again, these are writes.)

solution:
The answer to each question is given below.

1. IS on D; IS on F_2; IS on P_{1200}; S on P_{1200}:5.
2. IS on D; IS on F_2; IS on P_{1200}, S on 1201 through 1204, IS on $P_1$205; S on P_{1200}:98/99/100, S on P_{1205}:1/2.
3. IS on D; S on F_1.
4. IS on D; IS on F_1; S on P_{500} through P_{520}.
5. IS on D; S on F_1 (performance hit of locking 970 pages is likely to be higher than other blocked transactions).
6. IS and IX on D; SIX on F_1.
7. IX on D; IX on F_2; X on P_{1200}. (Locking the whole page is not necessary, but it would require some reorganization or compaction after the delete.)
8. IX on D; X on F_1 and F_2. (There are many ways to do this, there is a trade-off between overhead and concurrency.)