Data Storage and Retrieval
Scheduling Transactions: Practice Questions

Scheduling of transactions is under the control of a transaction manager and may involve different users submitting transactions that execute concurrently, for example accessing and updating same database records. The transaction manager is responsible for interleaving the different transactions safely.

In the following examples, the transaction operations that denote the beginning and end of transactions have not been applied. Can you identify the problems that occur when the transaction operations are omitted and identify which transaction operations should be applied to protect against or solve the issues?

Problem 1: Starting value x = 100

Time	S1	S2	Comments
t1	read_item(x)		S1 reads x
t2	X := x-40	read_item(x)	S1 processes the calculation to update x, but S2 can read x before S1 writes the newly calculated amount, so S2 can read the old value for X.
t3	write_item(x)	X := x+20	S1 writes its new value to x and the value of x is updated in the database. S2 performs its calculation on x, but it is the view of x that it read at t2. The S1 write is not committed here, and therefore it is a temporary value (until the change is committed), but S1 adding a commit here would not solve the problem. The problem arises due to the interleaving of processes that results in 2 processes doing different operations on their own view of x.
t4		write_item(x)	S2 writes its value for x and overwrites the x value calculated by S1.

Question 1: Will the view of the data be correct at the end of this sequence? If not, describe the sequence of events that results in an incorrect view of the data and where the error occurs. What is the final value of x?

Answer 1: The above interleaved operation will lead to an incorrect value for data item X because at time t2, S2 reads in the original value of X, which is before S1 changes it in the database, and hence the updated value resulting from S1 is lost. S1 reduces X by 40 and therefore, at time t3, 60 is written to the database. Time t4, S2 writes an updated version of X To the database but this is based on the original value of X. The final result should be X =

100 - 40 + 20 = 80; but in the concurrent operations in our example it is X equals 120 because the update at time t3 by S1 was lost. This is known as the lost update problem.

Problem 2:

Time	S1	S2	Comments
t1		read_item(x)	S2 reads x
t2		X := x-20	S2 calculates x-20
t3		write_item(x)	S2 writes the new value of x but it is a temporary value until the transaction is committed.
t4	read_item(x)		S1 reads the temporary value of x, but something happens that makes S2 abort the transaction.
t5		?	If S2 applies a ROLLBACK then S1 has an incorrect view of x. S2 has to COMMIT the new value of x before S1 reads x to make the data consistent.

Question 2: (a) Remembering that the transaction operations have not been applied, what could transaction operation would result in an inconsistent view of the data if applied at t5? (b) what transaction operation could be applied to fix the problem and when should it be applied?

Answer 2: This is an example of the uncommitted dependency problem. This occurs when a transaction is allowed to retrieve, or worse update, a record that has been updated by another transaction, but has not yet been committed by that other transaction. This is because if it has not yet been committed there is always a possibility that it will never be committed, and that it will be rolled back instead. If S2 applies rollback at time t5, S1 has already read the temporary incorrect value of X. In this example, S1 thinks that X has the value that S1 has temporarily updated at time t3. To solve this S2 must commit its write operation before S1 reads X at time t4.

Question 3:

Assume Situation in which there are three accounts that have the following values ACC1=40, ACC2=50, ACC3=30.

What are the account values, and the value of sum, after the following sequence of operations? What is incorrect?

Sum=0

Time	S1	S2	Comments
t1	read_item(ACC1)		40
t2	Sum := Sum + ACC1		Sum = 40
t3	read_item(ACC2)		50
t4	Sum := Sum + ACC2		Sum = 90
t5		read_item(ACC3)	30

t6		read_item(ACC1)	40
t7		ACC3 = ACC3 - 10	20
t8		ACC1 = ACC1 + 10	50
t9		write_item(ACC3)	ACC3 := 20
t10		write_item(ACC1)	ACC1 := 50
t11		COMMIT	All S2 operations are committed.
t12	read_item(ACC3)		20
t13	Sum := Sum + ACC3		Sum = 90 + 20 = 110 Again, this problem occurs due to interleaving. However, this time it is the aggregate function that is incorrect. S1 is trying to sum the account balances. At the start the balances sum to 120 and at the end the account balances sum to 120. However, S1 thinks the balances sum to 110.

Answer 3: This is known as the inconsistent analysis problem, it occurs when a transaction reads several values, but our second transaction updates some of these values during the execution of the 1st. This is significant, for example 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 functions may calculate some values before they are updated, and others after they are updated. Therefore, different summaries such as sums and counts, may be inconsistent. In this example, two transactions are operating on account records. Transaction S1 is summing account balances, transaction S2 is transferring an amount from account 3 to account one. The result produced by transaction S1 is incorrect. If S1 where to write that result back into the database it would leave the database in an inconsistent state. Transaction S1 has seen an inconsistent state of the database. And therefore performed an inconsistent analysis.