



DS Monsoon24

Quiz 4

1. Consider the three transactions below (only these transactions are simultaneously executing in the system) and four data items – a, b, c, and x. T1 {R1(a), R1(b), R1(c), $x=a+b+c$, print x} T2 {R1(b), R1(c), R1(a), $x= b+c+a$, print x} T3 {R1(c), R1(a), R1(b), $x= c+a+b$, print x} Select one or more:
- a. With time stamp ordering concurrency control algorithm, transaction T2 will always get aborted and restarted.
 - b. There is a serializable execution for these transactions. 
 - c. There is a serial execution for these transactions. 
 - d. With locking concurrency control algorithm, transaction T2 will always get deadlocked.
 - e. None of the others

Answer:

- b. There is a serializable execution for these transactions.
- c. There is a serial execution for these transactions.

Explanation:



In the given scenario, each transaction T1, T2, and T3 performs read operations (R1) on the data items a, b, and c, followed by a computation and a print statement. The order of operations within each transaction is fixed, and there is no conflicting write operation on the same data items.

Since the transactions do not have conflicting writes and the read operations are consistent within each transaction, a serializable execution is possible. Additionally, a serial execution is also possible, as the transactions can be executed in some order without violating the consistency of the final state.

The Time Stamp Ordering (TSO) concurrency control algorithm is designed to ensure serializability, so it does not lead to the abort and restart of T2 in this scenario.

The locking concurrency control algorithm is not explicitly mentioned in the question, and there is no indication of deadlocks occurring, so option d is not necessarily true.

Therefore, options b and c are correct.



2. Consider the three transactions below (only these transactions are simultaneously executing in the system) and four data items – a, b, c, and x. T1 {R1(a), R1(b), R1(c), $x = a + b + c$, W1(x)} T2 {R2(b), R2(c), R2(a), $x = b + c + a$, W2(x)} T3 {R3(c), R3(a), R3(b), $x = c + a + b$, W3(x)} Select one or more:
- a. With locking concurrency control algorithm, transaction T2 will always get deadlocked.
 - b. With time stamp ordering concurrency control algorithm, transaction T2 will get always aborted and restarted.
 - c. There is a serial execution for these transactions. 
 - d. None of the others
 - e. There is a serializable execution for these transactions. 

Answer:

- c. There is a serial execution for these transactions.
- e. There is a serializable execution for these transactions.

In this scenario, similar to the previous question, the transactions T1, T2, and T3 are accessing the data items a, b, c, and x in different orders. However, each transaction is performing a read and write on x based on the values read from a, b, and c. Since the final result involves a sum and there is no conflicting write operation on the same data item by different transactions, it is possible to find a serializable execution order.

Therefore, option c and e are correct.



3. Consider the three transactions below (only these transactions are simultaneously executing in the system) and four data items – a, b, c, and x. T1 {R1(a), R1(x), $a = a + x$, W1(a)} T2 {R2(b), R2(x), $b = b + x$, W2(b)} T3 {R3(c), R3(x), $c = c + x$, W3(c)} Select one or more:
- a. With a locking concurrency control algorithm, transaction T3 can get deadlocked.
 - b. There is a serializable execution for these transactions. 
 - c. There is a serial execution for these transactions. 
 - d. None of the others
 - e. With the timestamp ordering concurrency control algorithm, transaction T1 can get aborted and restarted.

Answer:

- b. There is a serializable execution for these transactions.
- c. There is a serial execution for these transactions.

In this scenario, each transaction T1, T2, and T3 is performing a read and write on a different data item (a, b, c) and the common data item x. The write operations are based on the values read from the corresponding data items and x. Since there is no conflicting write operation on the same data item by different transactions, it is possible to find a serializable execution order.

Therefore, option b,c are correct.

4. Consider the three transactions below (only these transactions are simultaneously executing in the system) and three data items – a, b, and c. T1 {R1(a), R1(b), $c=a+b$, W1(c)} T2 {R2(b), R2(c), $a = b+c$, W2(a)} T3 {R3(c), R3(a), $b=c+a$, W3(b)} The initial values of a, b, and c are 1, 2, and 3, respectively. Select one or more:
- a. None of the others
 - b. Locking will always cause deadlocks when executing these transactions.
 - c. There is no serializable execution of the above transactions. 
 - d. Among all possible serial executions of the above transactions, the maximum value for any of a, or b, or c is 13. 
 - e. The timestamp ordering algorithm can generate a serial execution for these transactions.




Answer:

- c. There is no serializable execution of the above transactions.
- d.. Among all possible serial executions of the above transactions, the maximum value for any of a, or b, or c is 13.

$T2 \rightarrow T3 \rightarrow T1$: T2 makes $a = 5$, T3 makes $b = 8$, T1 makes $c = 13$. Is the maximum value occurs in this order of execution

In this scenario, the transactions T1, T2, and T3 are performing read and write operations on the data items a, b, and c. The operations involve arithmetic calculations and updates to multiple data items in each transaction. As a result, there is no way to find a serializable execution order that would produce the same final values for a, b, and c as the concurrent execution.

Therefore, option c,d are correct

5. The fix and flush recovery algorithm Select one or more:
- a. Redos are not required during recovery.
 - b. Undos are not required during recovery. 
 - c. None of the others
 - d. It can cause more page replacements in the main memory. 
 - e. The same block may have to be written to the disk multiple times. 

Answer:

- b. Undos are not required during recovery:

In a fix-and-flush recovery algorithm, changes made by a transaction are not immediately reflected in the database. Instead, they are buffered in main memory. If a transaction needs to be undone during recovery, it can be simply ignored, and the

changes are not flushed to the disk. This means that the fix-and-flush recovery algorithm does not require explicit undo operations during the recovery process.

d. It can cause more page replacements in the main memory:

In a fix-and-flush recovery algorithm, updates made by a transaction are kept in main memory until the transaction commits. If the main memory is limited and there are many transactions with pending changes, it can lead to frequent page replacements in the main memory. This is because the updates must be applied to the corresponding database pages, and the available memory may not be sufficient to keep all the necessary pages in memory simultaneously.

e. The same block may have to be written to the disk multiple times:

Since updates are held in main memory until the transaction commits, the corresponding database pages may need to be written to the disk multiple times. This is because different transactions may modify the same block, and each transaction's changes are only flushed to the disk when it commits. Consequently, the same block may be written to the disk multiple times as different transactions complete.

Options a and c were not selected because redos may still be required during recovery, and the fix-and-flush algorithm does not inherently eliminate the need for redo operations. Additionally, "None of the others" (option c) was not selected because, while fix-and-flush recovery has certain characteristics, the statements chosen (b, d, e) are applicable to this type of recovery algorithm.