CS431 Quiz 3 Solutions

1. The consistency of transactions

   (a) refers to the fact that if a transaction is started in any database state, it leaves the database in a consistent state.

   (b) ensures that the outputs produced by the transactions will always be consistent.

   (c) implies that when a transaction does multiple updates, each will be consistent.

   (d) None of the above.

   Ans: (D)

2. Atomicity of transactions refers to the requirement that

   (a) each transaction updates exactly one indivisible entity of a database

   (b) either all operations of the transaction are reflected in the database or none are

   (c) each transaction preserves the correctness of the database

   (d) when one transaction executes all other transactions should be made to wait

   Ans: (B)

3. Under the 2-phase locking protocol

   (a) once a transaction has released a lock it can not acquire any more locks

   (b) a transaction can release all its locks when it commits

   (c) a transaction can acquire all the locks it needs when it begins

   (d) all of the above

   Ans: (D)

4. Given a schedule for transactions t_1 and t_2 we can say that

   (a) the schedule can be serialized if t_1 and t_2 resulted from the use of two-phase locking

   (b) the transactions compute the correct result if t_1 executed only after t_2 committed

   (c) the transactions compute the correct result if t_1 and t_2 executed concurrently but did not access any common data items

   (d) all of the above statements are correct

   Ans: (D)

5. A bank offers its account holders 7% interest if their balance is above Rs. 10000, else 6%. To achieve this, each of the following SQL statements is executed as separate transactions t_1 and t_2.

   t_1 : **update** *account* **set** *balance* = *balance* * 1.6
   where *balance* <= 10000
   t_2 : **update** *account* **set** *balance* = *balance* * 1.7
   where *balance* > 10000

   Then t_1 and t_2

   (a) must be executed one after the other but either order is acceptable

   (b) must be executed one after the other but with t_1 going first

   (c) must be executed one after the other but with t_2 going first

   (d) can be executed concurrently

   Ans: (C)

6. What is the serialization order of transactions when (a) two phase locking is used for concurrency control, (b) timestamp-based is used for concurrency control?
   Ans:
   In 2PL, serialization order is the order of the lock points.
   In timestamp-based concurrency control, serialization order is the order of transaction timestamps.

7. Suppose we log only "after images" of an updated page. Describe (a) what should happen at commit time (b) what should not happen during transaction execution time?
   Ans:
   At commit time:
   The log of all changes made by the transaction should be written to the disk.
   At transaction execution time:
   No page that is modified by a transaction when in the buffer cache should be allowed to be written to the disk before the transaction actually commits. This is because the log does not have before images and changes once made to the disk cannot be rolled back, if the transaction aborts.

8. A system caches *logical* objects, e.g., (parts of) base relations, intermediate results, as opposed to physical blocks. List 3 considerations involved in cache replacement.
   (Note that cache here refers to buffer cache)
   Ans: Issues in cache replacement:
   (a) Suppose a cached logical object consists of two different physical blocks, each of which is modified by a different transaction. Assume one of the transaction, say T, has committed and the other, say S, is still in progress. When the logical object needs to be evicted from the buffer cache due to cache replacement, we must ensure that the undo log of the uncommitted

transaction S should be written to the disk. (The redo log for the committed transaction has already been written to the disk at commit time).

(b) The cache replacement policy should take into account the number of active transactions that used the logical object in the recent past.

9. For better performance, a database is replicated on multiple sites. Suppose a transaction executing from a mobile laptop visits different replicas and reads and writes (parts of) the database.

   Design a protocol whereby a transaction can read *globally-consistent* data from the database replica "nearest" to it. In particular, what should a reading transaction lock? what should a writing transaction lock?

   Will your transaction lead to deadlocks? If so, how can you avoid it?
   Ans:
   A reading transaction should lock the required data item only on that database replica that is "nearest" to it.
   A writing transaction should lock the required data item on all the database replica.

   This can lead to deadlock. Deadlock can be avoided by ordering the locks acquired on the database replicas for the write. This can be done by imposing the constraint that any writing transaction should lock and update the database replica in a fixed order (which is the same for each transaction).

10. One way to reduce transaction conflicts is to "chop" a transaction into many components: together the chopped components produce the same (consistent) updates and outputs as the original version, but each uses less locks.

    Argue about the correctness of the following chopping heuristic: "Suppose transaction T accesses X and Y, but any other transaction S accesses at most one of X or Y and nothing else. Then, T can be chopped into two components, one of which accesses X and the other accesses Y."
    Ans:
    The chopping heuristic is correct. This can be argued as follows:
    Consider T is chopped into T_1 and T_2. Assume T_1 accesses only X and T_2 accesses only Y. (The other cases can be proved using a similar argument). Further, if we assume that S accesses X only, then any schedule produced by T_1, T_2 and S is serializable (i.e. any schedule is equivalent to either S, T or T, S.