Question 1

b) 1. Lost Updates

The lost update problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost (overwritten by the other transaction). Consider the following PRODUCT table example.

One of the PRODUCT table’s attributes is a product’s quantity on hand (PROD\_QOH).

Assume that you have a product whose current PROD\_QOH value is 35. Also assume that two concurrent transactions, T1 and T2, occur that update the PROD\_QOH value for some item in the PRODUCT table.

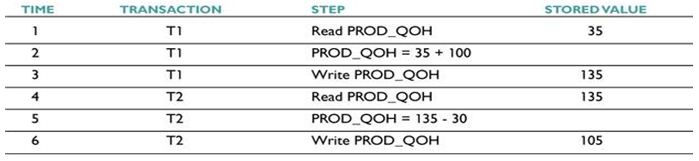
The transactions are as follows.

Two concurrent transactions update PROD\_QOH:

    Transaction                        Operation

T1: Purchase 100 units         PROD\_QOH = PROD\_QOH + 100  
T2: Sell 30 units                 PROD\_QOH = PROD\_QOH – 30

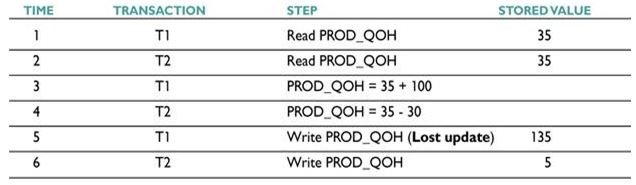
The Following table shows the serial execution of those transactions under normal circumstances, yielding the correct answer PROD\_QOH = 105.



But suppose that a transaction is able to read a product’s PROD\_QOH value from the table before a previous transaction (using the same product) has been committed.

The sequence depicted in the following Table shows how the lost update problem can arise.

Note that the first transaction (T1) has not yet been committed when the second transaction (T2) is executed. Therefore, T2 still operates on the value 35, and its subtraction yields 5 in memory. In the meantime, T1 writes the value 135 to disk, which is promptly overwritten by T2. In short, the addition of 100 units is “lost” during the process.



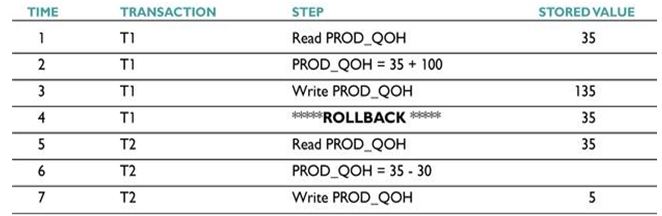
2. Uncommitted Data

The phenomenon of uncommitted data occurs when two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data—thus violating the isolation property of transactions.

To illustrate that possibility, let’s use the same transactions described during the lost update’s discussion. T1 has two atomic parts to it, one of which is the update of the inventory, the other possibly being the update of the invoice total (not shown). T1 is forced to roll back due to an error during the updating of the invoice’s total; hence, it rolls back all the way, undoing the inventory update as well. This time, the T1 transaction is rolled back to eliminate the addition of the 100 units. Because T2 subtracts 30 from the original 35 units, the correct answer should be 5.

    Transaction                Operation  
T1: Purchase 100 units     PROD\_QOH = PROD\_QOH + 100 (Rolled back)  
T2: Sell 30 units                 PROD\_QOH = PROD\_QOH – 30

The following Table shows how, under normal circumstances, the serial execution of those transactions yields the correct answer.



3. Inconsistent Retrievals

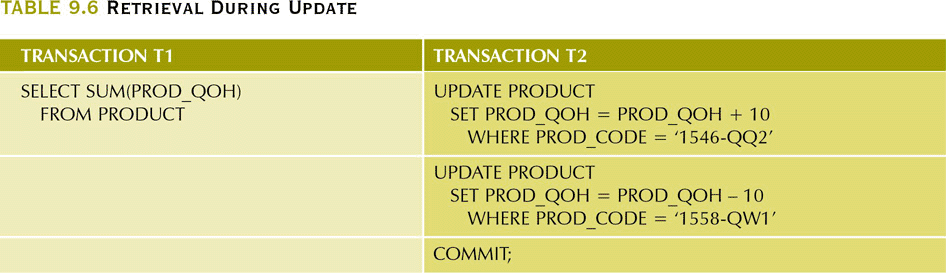
Inconsistent retrievals occur when a transaction accesses data before and after another transaction(s) finish working with such data. For example, an inconsistent retrieval would occur if transaction T1 calculated some summary (aggregate) function over a set of data while another transaction (T2) was updating the same data. The problem is that the transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

To illustrate that problem, assume the following conditions:

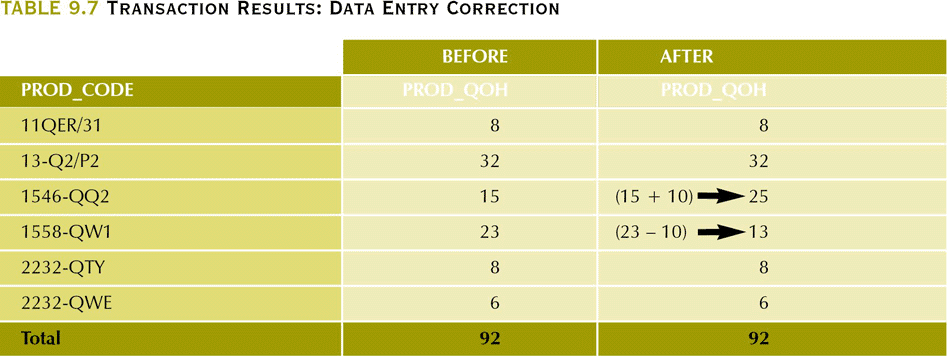
1. T1 calculates the total quantity on hand of the products stored in the PRODUCT table.

2. At the same time, T2 updates the quantity on hand (PROD\_QOH) for two of the PRODUCT table’s products.

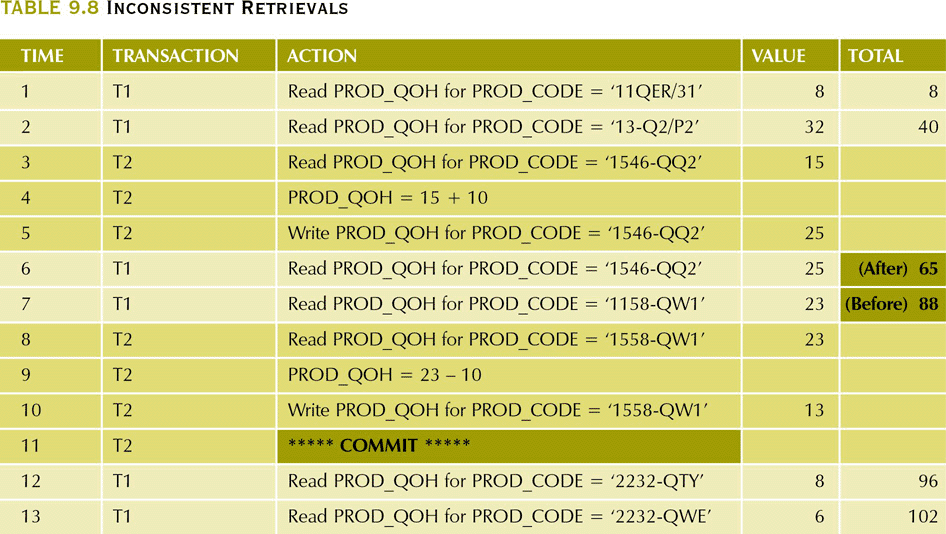
The two transactions are shown in the following Table:



While T1 calculates the total quantity on hand (PROD\_QOH) for all items, T2 represents the correction of a typing error: the user added 10 units to product 1558-QW1’s PROD\_QOH but meant to add the 10 units to product 1546-QQ2’s PROD\_QOH. To correct the problem, the user adds 10 to product 1546-QQ2’s PROD\_QOH and subtracts 10 from product 1558-QW1’s PROD\_QOH. The initial and final PROD\_QOH values are reflected in the following Table



The following table demonstrates that inconsistent retrievals are possible during the transaction execution, making the result of T1’s execution incorrect. The “After” summation shown in Table 10.9 reflects the fact that the value of 25 for product 1546-QQ2 was read after the WRITE statement was completed. Therefore, the “After” total is 40 + 25 = 65. The “Before” total reflects the fact that the value of 23 for product 1558-QW1 was read before the next WRITE statement was completed to reflect the corrected update of 13. Therefore, the “Before” total is 65 + 23 = 88



The computed answer of 102 is obviously wrong because you know from the previous Table that the correct answer is 92. Unless the DBMS exercises concurrency control, a multiuser database environment can create havoc within the information system.

c)

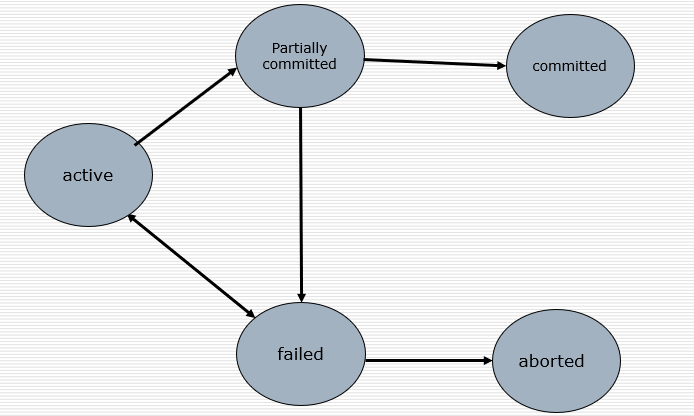
**Active** –the initial state; the transaction stays in this state while it is executing.

**Partially committed** –after the final statement has been executed.

**Failed** –after the discovery that normal execution can no longer proceed.

**Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction.

**Committed** – after successful completion.

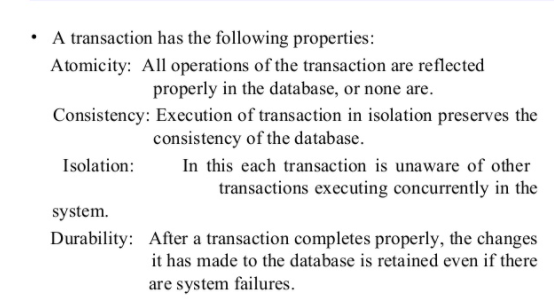


Question 2

ai)

1. A transaction is a logical unit of work. Each transaction is a sequence of logically related commands that accomplish one task and transform the database from one consistent state into another
2. Any action that reads from and/or writes to a database may consist of SELECT UPDATE or INSERT, a logical unit of work that must be either entirely completed or aborted

b)

Serializability: Ensures that the concurrent execution of several transactions yields consistent results. Multiple, concurrent transactions appear as if they executed in a serial order (one after another)

Notes:

