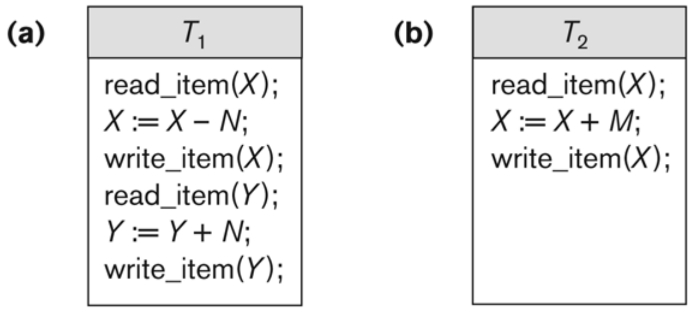
CHAPTER 17: INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS AND THEORY

Answers to Selected Exercises

17.14 Change transaction T 2 in Figure 17.2b to:

```
read_item(X);
X:= X+M;
if X > 90 then exit
else write_item(X);
```

Discuss the final result of the different schedules in Figure 17.3(a) and (b) where M = 2 and N = 2, with respect to the following questions. Does adding the above condition change the final outcome? Does the outcome obey the implied consistency rule (that the capacity of X is 90)?



(a)

```
T1
read_item(X);
X := X − N;
write_item(X);
read_item(Y);
Y := Y + N;
write_item(Y);
```

(b)

```
T2
read_item(X);
X := X + M;
write_item(X);
```

Figure 17.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

Answer:

The above condition does not change the final outcome unless the initial value of X > 88. The outcome, however, does obey the implied consistency rule that X < 90, since the value of X is not updated if it becomes greater than 90.

17.15 Repeat Exercise 17.14 adding a check in T 1 so that Y does not exceed 90.

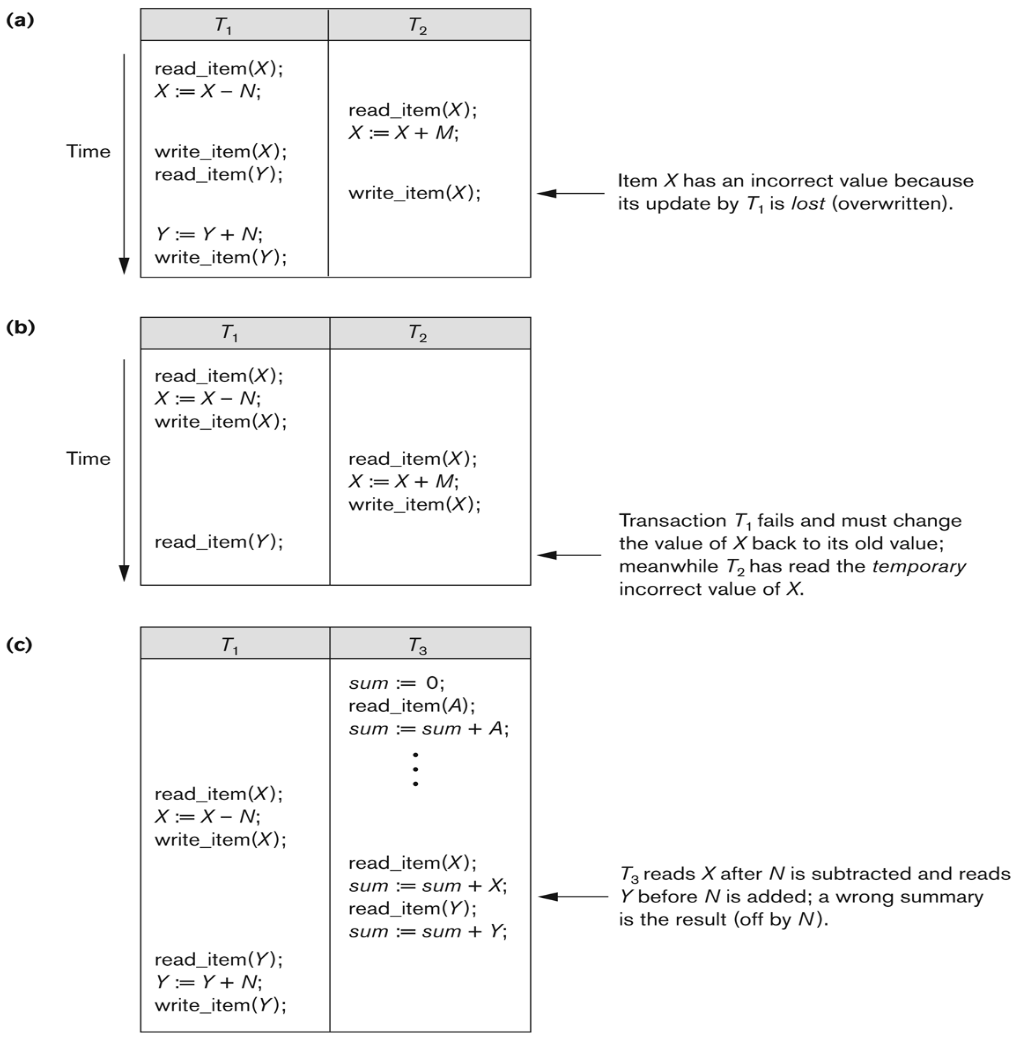
Answer:

T1	T2

read_item(X);
X := X - N
write_item(X);
read_item(Y);
Y := Y + N;
if Y> 90 then exit
write_item(Y);
	read_item(X);
	X := X + M;
	if X> 90 then exit
	write_item(X);

The above condition does not change the final outcome unless the initial value of X > 88 or Y > 88. The outcome obeys the implied consistency rule that X < 90 and Y < 90.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.


(a)

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;

write_item(X);
read_item(Y);	

	write_item(X);
$Y := Y + N$; write_item(Y);	

(b)

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);	

(c)

T_1	T_3
	$sum := 0$; read_item(A); $sum := sum + A$; ⋮

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$sum := sum + X$;
	read_item(Y);
	$sum := sum + Y$;

read_item(Y);
$Y := Y + N$;
write_item(Y);	

Time
Item X has an incorrect value because its update by T_1 is *lost* (overwritten).
Time
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.
T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

17.16 Add the operation commit at the end of each of the transactions T 1 and T 2 from Figure 17.2; then list all possible schedules for the modified transactions.

Answer:

T1	T2

read_item(X);
X := X - N
write_item(X);
read_item(Y);
Y := Y + N;
write_item(Y);
commit T 1
	read_item(X);
	X := X + M;
	write_item(X);
	commit T 2

The transactions can be written as follows using the shorthand notation:
T 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ;
T 2 : r 2 (X); w 2 (X); C 2 ;

In general, given m transactions with number of operations n1, n2, ..., nm, the number of possible schedules is: (n1 + n2 + ... + nm)! / (n1! * n2! * ... * nm!), where ! is the factorial function. In our case, m =2 and n1 = 5 and n2 = 3, so the number of possible schedules is:

(5+3)! / (5! * 3!) = 8*7*6*5*4*3*2*1/ 5*4*3*2*1*3*2*1 = 56.

Below are some possible schedules:
S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; r 2 (X); w 2 (X); C 2 ;
S 2 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ;
S 3 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 1 ; C 2 ;
S 4 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 2 ; C 1 ;
S 5 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); C 1 ; w 2 (X); C 2 ;
S 6 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 1 ; C 2 ;
S 7 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 2 ; C 1 ;
S 8 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 1 ; C 2 ;
S 9 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ;
S 10 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); C 2 ; w 1 (Y); C 1 ;
S 11 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ;

17.17 List all possible schedules for transactions T 1 and T 2 from figure 17.2, and determine which is conflict serializable (correct) and which are not.

Answer:

T1	T2

read_item(X);
X := X - N
write_item(X);
read_item(Y);
Y := Y + N;
write_item(Y);
	read_item(X);
	X := X + M;
	write_item(X);

The transactions can be written as follows using shorthand notation:
T 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y);
T 2 : r 2 (X); w 2 (X);

In this case, m =2 and n1 = 4 and n2 = 2, so the number of possible schedules is:
(4+2)! / (4! * 2!) = 6*5*4*3*2*1/ 4*3*2*1*2*1 = 15.

Below are the 15 possible schedules, and the type of each schedule:

Notes:

- Serial schedule:
  - A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule. Otherwise, the schedule is called nonserial schedule.
- Serializable schedule:
  - A schedule S is serializable if it **is equivalent** to some serial schedule of the same n transactions.
  -

- Result equivalent: Two schedules are called result equivalent if they produce the same final state of the database.
- Conflict equivalent: Two schedules are said to be conflict equivalent if <u>the order of any **two conflicting operations**</u> is the **same** in both schedules.
- Conflict serializable: A schedule S is said to be conflict serializable if it is <u>conflict equivalent</u> to some <u>serial schedule S'</u>.

S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); serial (and hence also serializable)
S 2 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); (conflict) serializable
S 3 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); (conflict) serializable
S 4 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); (conflict) serializable
S 5 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); (conflict) serializable
S 6 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); (conflict) serializable
S 7 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable
S 8 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable
S 9 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 10 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 11 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable
S 12 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable
S 13 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 14 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 15 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); serial (and hence also serializable)

17.18 How many serial schedules exist for the three transactions in Figure 17.8 (a)? What are they? What is the total number of possible schedules?

Figure 17.8
Another example of serializability testing. (a) The read and write operations of three transactions T_1, T_2, and T_3. (b) Schedule E. (c) Schedule F.

(a)

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

(b)

Time

Transaction T_1	Transaction T_2	Transaction T_3

	read_item(Z);
	read_item(Y);
	write_item(Y);	

		read_item(Y);
		read_item(Z);

read_item(X);
write_item(X);		

		write_item(Y);
		write_item(Z);

	read_item(X);	

read_item(Y);
write_item(Y);		

	write_item(X);	

Schedule E

(c)

Time

Transaction T_1	Transaction T_2	Transaction T_3

		read_item(Y);
		read_item(Z);

read_item(X);
write_item(X);		

		write_item(Y);
		write_item(Z);

	read_item(Z);	

read_item(Y);
write_item(Y);		

	read_item(Y);
	write_item(Y);
	read_item(X);
	write_item(X);	

Schedule F

Partial Answer:

T1 T2 T3
T3 T2 T1
T2 T3 T1
T2 T1 T3
T3 T1 T2
T1 T3 T2
Total number of serial schedules for the three transactions = 6
In general, the number of serial schedules for n transactions is n! (i.e. factorial(n))

17.20 Why is an explicit transaction end statement needed in SQL but not an explicit begin statement?

Answer:

A transaction is an atomic operation. It has only one way to begin, that is with "Begin Transaction" command but it could end up in two ways: Successfully installs its updates to the database (i.e., commit) or Removes its partial updates (which may be incorrect) from the database (abort). Thus, it is important for the database systems to identify the right way of ending a transaction. It is for this reason an "End" command is needed in SQL2 query.

17.22 Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

(a) r1 (X); r3 (X); w1(X); r2(X); w3(X)

(b) r1 (X); r3 (X); w3(X); w1(X); r2(X)

(c) r3 (X); r2 (X); w3(X); r1(X); w1(X)

(d) r3 (X); r2 (X); r1(X); w3(X); w1(X)

<u>Answer:</u>

Let there be three transactions T1, T2, and T3. They are executed concurrently and produce a schedule S. S is serializable if it can be reproduced as at least one serial schedule (T1 T2 T3 or T1 T3 T2 or T2 T1 T3 or T2 T3 T1 or T3 T1 T2 or T3 T2 T1).

(a) This schedule is not serializable because T1 reads X (r1(X)) before T3 but T3 reads X(r3(X)) before T1 writes X (w1(X)), where X is a common data item. The operation r2(X) of T2 does not affect the schedule at all so its position in the schedule is irrelevant. In a serial schedule T1, T2, and T3, the operation w1(X) comes after r3(X), which does not happen in the question.

(b) This schedule is not serializable because T1 reads X (r1(X)) before T3 but T3 writes X(w3(X)) before T1 writes X (w1(X)). The operation r2(X) of T2 does not affect theschedule at all so its position in the schedule is irrelevant. In a serial schedule T1, T3, and T2, r3(X) and w3(X) must come after w1(X), which does not happen in the question.

(c) This schedule is serializable because all conflicting operations of T3 happens before all conflicting operation of T1. T2 has only one operation, which is a read on X (r2(X)), which does not conflict with any other operation. Thus this serializable schedule is equivalent to r2(X); r3(X); w3(X); r1(X); w1(X) (e.g., T2 T3 T1) serial schedule.

(d) This is not a serializable schedule because T3 reads X (r3(X)) before T1 reads X (r1(X)) but r1(X) happens before T3 writes X (w3(X)). In a serial schedule T3, T2, and T1, r1(X) will happen after w3(X), which does not happen in the question.

17.23 Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializibility (precedence) graphs for S1 and S2 and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).

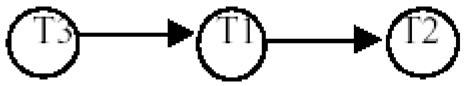
T1: r1(x); r1(z); w1(x)
T2: r2(z); r2(y); w2(z); w2(y)
T3: r3(x); r3(y); w3(y)

S1: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y)
S2: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y)

Answers:

Schedule S1: It is a serializable schedule because

- T1 only reads X (r1(X)), which is not modified either by T2 or T3,
- T3 reads X (r3(X)) before T1 modifies it (w1(X)),
- T2 reads Y (r2(Y)) and writes it (w2(Y)) only after T3 has written to it (w3(Y))
- Thus, the serializability graph is



Schedule is not a serializable schedule because

- T2 reads Y (r2(Y)), which is then read and modified by T3 (w3(Y))
- T3 reads Y (r3(Y)), which then modified before T2 modifies Y (w2(Y))

In the above order T3 interferes in the execution of T2, which makes the schedule nonserializable.