CHAPTER 20: INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS and THEORY

Answers to Selected Exercises

20.14 - Change transaction T 2 in Figure 21.2b to read:
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 21.3 (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)?

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.

20.15 - Repeat Exercise 21.14 adding a check in T 1 so that Y does not exceed 90.

Answer:
T1 T2
read_item(X);
X := X-N;
read_item(X);
X := X+M;
write_item(X);
read_item(Y);
if X > 90 then exit
else write_item(X);
Y := Y+N;
if Y> 90 then
exit
else write_item(Y);
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.

20.16 - Add the operation commit at the end of each of the transactions T 1 and T 2 from Figure 21.2; then list all possible schedules for the modified transactions.
Determine which of the schedules are recoverable, which are cascadeless, and which are strict.

Answer:
T 1 T 2
read_item(X); read_item(X);
X := X - N X := X + M;

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

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 the 56 possible schedules, and the type of each schedule:
S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; r 2 (X); w 2 (X); C 2 ; strict (and hence cascadeless)
S 2 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ; recoverable
S 3 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 1 ; C 2 ; recoverable
S 4 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 2 ; C 1 ; non-recoverable
S 5 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); C 1 ; w 2 (X); C 2 ; recoverable
S 6 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 1 ; C 2 ; recoverable
S 7 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 2 ; C 1 ; non-recoverable
S 8 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 1 ; C 2 ; recoverable
S 9 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 10 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 11 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; recoverable
S 12 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; recoverable
S 13 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; non-recoverable
S 14 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; recoverable
S 15 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 16 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 17 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; recoverable
S 18 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 19 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 20 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; non-recoverable
S 21 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; strict (and hence cascadeless)
S 22 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; cascadeless
S 23 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; cascadeless
S 24 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; cascadeless
S 25 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; cascadeless
S 26 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; cascadeless
S 27 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 28 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 29 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 30 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 31 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 32 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 33 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 34 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 35 : r 1 (X); r 2 (X); w 2 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; strict (and hence

cascadeless)
S 36 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; strict (and hence cascadeless)
S 37 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; cascadeless
S 38 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; cascadeless
S 39 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; cascadeless
S 40 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; cascadeless
S 41 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; cascadeless
S 42 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 43 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 44 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 45 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 46 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 47 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 48 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 49 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 50 : r 2 (X); r 1 (X); w 2 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 51 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; non-recoverable
S 52 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; recoverable
S 53 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; recoverable
S 54 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; recoverable
S 55 : r 2 (X); w 2 (X); r 1 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; recoverable
S 56 : r 2 (X); w 2 (X); C 2 ; r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; strict (and hence cascadeless)

20.17 - List all possible schedules for transactions T 1 and T 2 from figure 21.2, and determine which are conflict serializable (correct) and which are not.

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

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:
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)

20.18 - How many *serial* schedules exist for the three transactions in Figure 21.8 (a)? What are they? What is the total number of possible schedules?

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))

20.19 - No solution provided.

20.20 - 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.

20.21 Describe situations where each of the different isolation levels would be useful for transaction processing.

Answer:
Transaction isolation levels provide a measure of the influence of other concurrent transactions on a given transaction. This affects the level of concurrency, that is, the level of concurrency is the highest in Read Uncommitted and the lowest in Serializable.

Isolation level Serializable: This isolation level preserves consistency in all situations, thus it is the safest execution mode. It is recommended for execution environment where every update is crucial for a correct result. For example, airline reservation, debit credit, salary increase, and so on.

Isolation level Repeatable Read: This isolation level is similar to Serializable except Phantom problem may occur here. Thus, in record locking (finer granularity), this isolation level must be avoided. It can be used in all types of environments, except in the environment where accurate summary information (e.g., computing total sum of a all different types of account of a bank customer) is desired.

Isolation level Read Committed: In this isolation level a transaction may see two different values of the same data items during its execution life. A transaction in this level applies write lock and keeps it until it commits. It also applies a read (shared) lock but the lock is released as soon as the data item is read by the transaction. This isolation level may be used for making balance, weather, departure or arrival times, and so on.

Isolation level Read Uncommitted: In this isolation level a transaction does not either apply a shared lock or a write lock. The transaction is not allowed to write any data item, thus it may give rise to dirty read, unrepeatable read, and phantom. It may be used in the environment where statistical average of a large number of data is required.

20.22 - 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)

Answer:
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 the schedule 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.

20.23 - 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(x); 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)

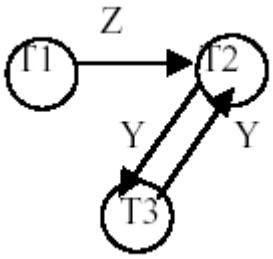
Answer:
Schedule S1: It is a serializable schedule because

- □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.



20.24 - Consider schedules S3, S4, and S5 below. Determine whether each schedule is strict, cascadeless, recoverable, or nonrecoverable. (Determine the strictest recoverability condition that each schedule satisfies.)

S3: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); c1; w3(y); c3; r2(y); w2(z); w2(y);c2
S4: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y); c1; c2; c3;
S5: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y); c3; c2;

Answer:
Strict schedule: A schedule is strict if it satisfies the following conditions:
1. Tj reads a data item X ***after*** Ti has written to X and Ti is terminated (aborted or committed)
2. Tj writes a data item X ***after*** Ti has written to X and Ti is terminated (aborted or committed)

Schedule S3 is not strict because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X)) but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.
Schedule S4 is not strict because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X)) but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.
Schedule S5 is not strict because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X)) but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.
Cascadeless schedule: A schedule is cascadeless if the following condition is satisfied:
□□Tj reads X only ***after*** Ti has written to X and terminated (aborted or committed).
Schedule S3 is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.
Schedule S4 is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.
Schedule S5 is ***not cascadeless*** because T3 reads X (r3(X)) ***before*** T1 commits or T2 reads
Y (r2(Y)) ***before*** T3 commits.

NOTE: According to the definition of cascadeless schedules S3, S4, and S4 are not cascadeless. However, T3 is not affected if T1 is rolled back in any of the schedules, that is, T3 does not have to roll back if T1 is rolled back. The problem occurs because these schedules are not serializable.

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:
□□Tj commits after Ti if Tj has read any data item written by Ti.
NOTE: Ci > Cj means Ci happens ***before*** Cj. Ai denotes abort Ti. To test if a schedule is recoverable one has to include abort operations. Thus in testing the recoverability abort operations will have to used in place of commit one at a time. Also the strictest condition is where a transaction neither reads nor writes to a data item, which was written to by a transaction that has not committed yet.

□□If A1>C3>C2, then S3 is ***recoverable*** because rolling back of T1 does not affect T2 and T3. If C1>A3>C2. S3 is ***not recoverable*** because T2 read the value of Y (r2(Y)) ***after*** T3 wrote X (w3(Y)) and T2 committed but T3 rolled back. Thus, T2 used non- existent value of Y. If C1>C3>A3, then S3 is ***recoverable*** because roll back of T2 does not affect T1 and T3. Strictest condition of S3 is C3>C2.
□□If A1>C2>C3, then S4 is ***recoverable*** because roll back of T1 does not affect T2 and T3. If C1>A2>C3, then S4 is ***recoverable*** because the roll back of T2 will restore the value of Y that was read and written to by T3 (w3(Y)). It will not affect T1. If C1>C2>A3, then S4 is ***not recoverable*** because T3 will restore the value of Y which was not read by T2. Strictest condition of S4 is C3>C2, but it is not satisfied by S4.
□□If A1>C3>C2, then S5 is ***recoverable*** because neither T2 nor T3 writes to X, which is written by T1. If C1>A3>C2, then S5 is ***not recoverable*** because T3 will restore the value of Y, which was not read by T2. Thus, T2 committed with a non-existent value of Y. If C1>C3>A2, then S5 is ***recoverable*** because it will restore the value of Y to the value, which was read by T3. Thus, T3 committed with the right value of Y. Strictest condition of S3 is C3>C2, but it is not satisfied by S5.

20.24 – No solution provided.