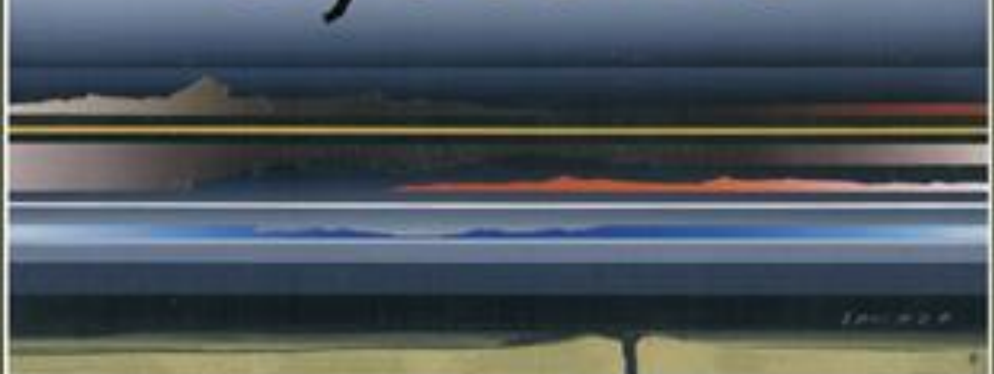
Fundamentals of

Database Systems

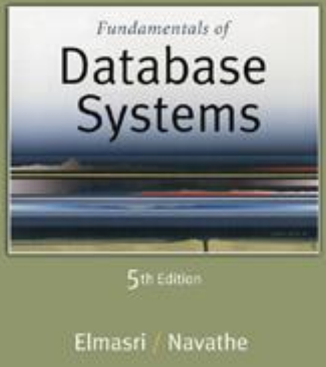


5th Edition

Elmasri / Navathe

Chapter 17

Introduction to Transaction Processing Concepts and Theory 2



Chapter Outline

1 Introduction to Transaction Processing

2 Transaction and System Concepts

3 Desirable Properties of Transactions

4 Characterizing Schedules based on Recoverability

5 Characterizing Schedules based on Serializability

6 Transaction Support in SQL

3 Desirable Properties of Transactions (1)

ACID properties:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another.
- **Isolation**: A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary (see Chapter 21).
- **Durability or permanency**: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

3 Desirable Properties of Transactions (1)

It's the responsibility of the:

- Transaction recovery subsystem of DBMS to ensure atomicity.
- Programmer is responsible for preserving of consistency or the DBMS module of that enforce integrity constraint.
- Concurrency control subsystem is responsible for isolation property
- Recovery subsystem of DBMS is responsible of the durability property.

Recoverability

4 Characterizing Schedules based on Recoverability (1)

- **Transaction schedule or history**:
  - When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction **schedule** (or history).

- A **schedule** (or **history**) S of n transactions T1, T2, …, Tn:
  - It is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in S, the operations of T1 in S must appear in the same order in which they occur in T1.
  - Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in S.

Characterizing Schedules based on Recoverability (2)

First, **once a transaction T is committed , it should never be necessary to rollback** T. This ensure that the durability property of a transaction is not violated.

The schedule that theoretically meet this criteria are called **recoverable schedule**; those that do not are called **nonrecoverable** and hence should not be permitted by the DBMS.

Characterizing Schedules based on Recoverability (2)

Example:

- When T_1 abort , I need to make undo for all T_2 operations , which is IMPOSSIBLE as it already **committed** so for durability to be satisfied its impossible to cancel the commit.
- So, the problem occurred because T_2 made commit before T_1 commits,

Schedule S1
R 1 (x)
W 1 (x)
R 2 (x)
C 2 (x)
A 1

SO, this is called **UNRECOVERABLE SCHEDULE** as I cannot cancel the changed that T_2 done.

- Solution→ T_2 can't make commit before T_1 commit

Example

Schedule S1	Schedule S2
R 5 (y)	R 5 (y)
W 5 (y)	W 5 (y)
R 1 (y)	R 1 (y)
W 1 (y)	W 1 (y)
C 1	A 5
C 5	

S1 (Its **unrecoverable** as T1 made commit before T5,
To be recoverable C5 first then C1)
S2 (**recoverable** , have no problem as no commit occurred)

Example

S_c: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; c_2; a_1

S_d: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$;$w_1(Y)$; c_1; c_2

S_e: $r_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$; a_1; a_2

S_c (Its unrecoverable as T_2 made commit before T_1)
S_d (recoverable , have no problem as T_1 commit before T_2)
S_e (recoverable as no problem as there is no commit , its abort)

Who wrote first should make commit first to be recoverable.

Characterizing Schedules based on Recoverability (2)

Schedules classified on recoverability:

- **Recoverable schedule**:
  - One where no transaction needs to be rolled back.
  - A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed.
- **Cascadeless schedule**:
  - One where every transaction **reads only** the items that are written by committed transactions.

Characterizing Schedules Based on Recoverability (cont.)

Example:

- Schedule B below is **not cascadeless** because T2 reads the value of X that was written by T1 before T1 commits
- If T1 aborts (fails), T2 must also be aborted (rolled back) resulting in cascading rollback
- To make it **cascadeless**, the r2(X) of T2 must be delayed until T1 commits (or aborts and rolls back the value of X to its previous value) – see Schedule C

- Schedule B: r1(X); w1(X); r2(X); w2(X); r1(Y); w1(Y); c1 (or a1);
- Schedule C: r1(X); w1(X); r1(Y); w1(Y); c1; r2(X); w2(X);

Characterizing Schedules based on Recoverability (3)

Schedules classified on recoverability (contd.):

- **Schedules requiring cascaded rollback**:
  - A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.
- **Strict Schedules**:
  - A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

Characterizing Schedules Based on Recoverability (cont.)

Example:

- Schedule C below is cascadeless and also strict (because it has no blind writes)
- Schedule D is cascadeless, but not strict (because of the blind write w3(X), which writes the value of X before T1 commits)
- To make it strict, w3(X) must be delayed until after T1 commits – see Schedule E

- **Schedule C: r1(X); w1(X); r1(Y); w1(Y); c1; r2(X); w2(X);**
- **Schedule D: r1(X); w1(X); w3(X); r1(Y); w1(Y); c1; r2(X); w2(X);**
Schedule E: r1(X); w1(X); r1(Y); w1(Y); c1; w3(X); r2(X); w2(X);

Serializability

5 Characterizing Schedules based on Serializability (1)

Serial Schedule:

A schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions. Otherwise, the schedule is called non-serial.

(a)

Time

T_1	T_2
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);

Schedule A

(b)

Time

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

Schedule B

5 Characterizing Schedules based on Serializability (1)

Non serial schedule: A schedule where the operations from a set of concurrent transactions are interleaved.

(c)

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);	

Time ↓

Schedule C

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

Time ↓

Schedule D

5 Characterizing Schedules based on Serializability (1)

- The problem of serial (not practical) they limit concurrency, so a delay occur.
- We need to go with the non serial schedule that avoid problems while executing as the serial, called
- Serializable schedule:
  - A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.
  - When we say S is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule which is considered correct.

Characterizing Schedules based on Serializability (2)

When are the two schedules considered equivalent?

- 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 the order of any two conflicting operations is the same in both schedules.
- Conflict serializable:
  - A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S’.

Characterizing Schedules based on Serializability (2)

When are the two schedules considered equivalent?

- **Result equivalent**:
  - Two schedules are called result equivalent if they produce the same final state of the database

S1	S2
Read_item (x);	Read_item (x);
x=x+10;	x=x*1.1
Write_item (x)	Write_item (x);

  - Two schedules are result equivalent for the initial value of x=100 but are not result equivalent in general, for ex: for x=500

So result equivalent is not practical

Characterizing Schedules based on Serializability (2)

Conflict equivalent: Two schedules are conflict equivalent if the relative order of any two conflicting operations is the same in both schedules.

Two operations are conflicting if:

- They are from different transactions
- They access the same data item X
- At least one is a write operation

- Read-Write conflict example: r1(X) and w2(X)
- Write-write conflict example: w1(Y) and w2(Y)

S1	
T1	T2
--	---
W(x)	
	R(x)

S2	
T1	T2
W(x)	---
	R(x)

S3	
T1	T2

R(x)	
	W(x)

Characterizing Schedules based on Serializability (2)

Conflict Serializability

- If a schedule S can be transformed into a schedule S´ by a series of swaps of non-conflicting instructions, we say that S and S´ are conflict equivalent.

- We say that a schedule S is conflict serializable if it is <u>conflict equivalent to a serial schedule</u>

Characterizing Schedules based on Serializability (2)

Conflict Serializability

Schedule 1 can be transformed into Schedule 2, a serial schedule where T2 follows T1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Conflict Serializability(cont.)

T1	T2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializability(cont.)

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializability(cont.)

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

Conflict Serializability(cont.)

T1	T2
R(A)	
W(A)	
R(A)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Serial Schedule

Characterizing Schedules based on Serializability (11)

Testing for conflict serializability: Algorithm 17.1:

- Looks at only read_Item (X) and write_Item (X) operations
- Constructs a precedence graph (serialization graph) - a graph with directed edges
- An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
- The schedule is serializable if and only if the precedence graph has no cycles.

Example of serializability Testing

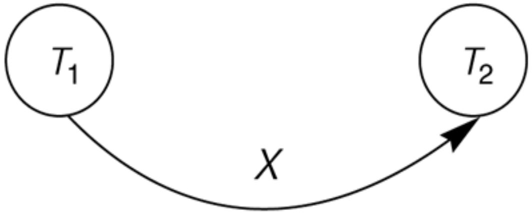
(a)

Time

T_1	T_2
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);

Schedule A

(a)



Example of serializability Testing

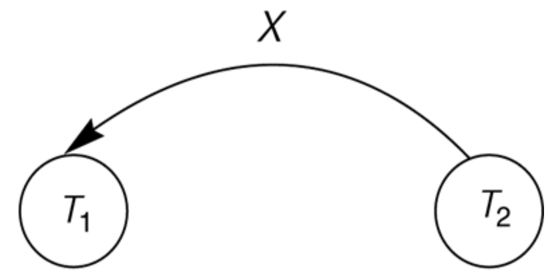
(b)

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

Time

Schedule B

(b)



Example of serializability Testing

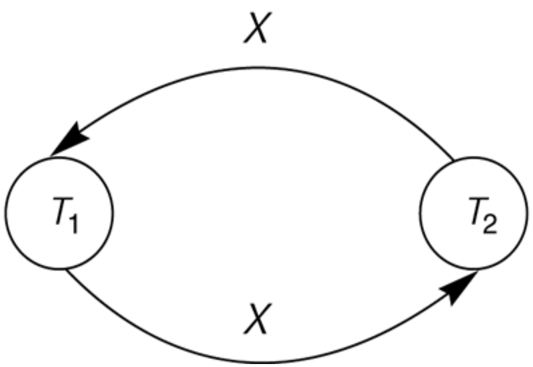
(c)

Time

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);	

Schedule C

(c)



Example of serializability Testing

Time

T_1	T_2

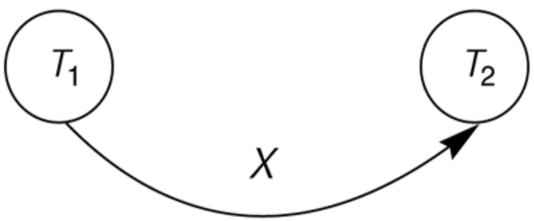
read_item(X);
$X := X - N$;
write_item(X);	

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

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

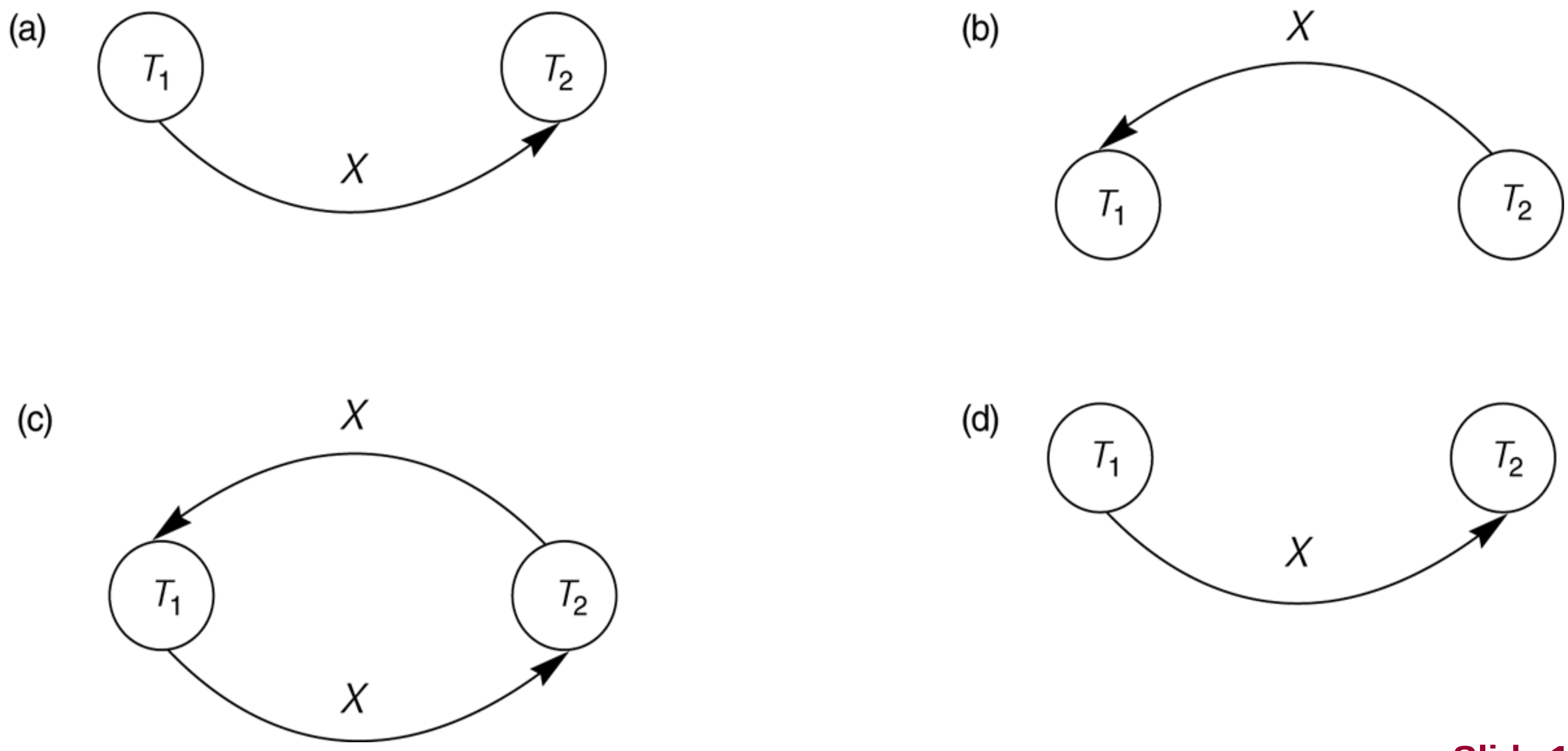
Schedule D

(d)



Constructing the Precedence Graphs

- FIGURE 17.7 Constructing the precedence graphs for schedules A and D from Figure 17.5 to test for conflict serializability.
  - (a) Precedence graph for serial schedule A.
  - (b) Precedence graph for serial schedule B.
  - (c) Precedence graph for schedule C (not serializable).
  - (d) Precedence graph for schedule D (serializable, equivalent to schedule A).



Another example of serializability Testing

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);

Another Example of Serializability Testing

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.

(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

Another Example of Serializability Testing

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*.

(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

Summary

- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability
- Transaction Support in SQL