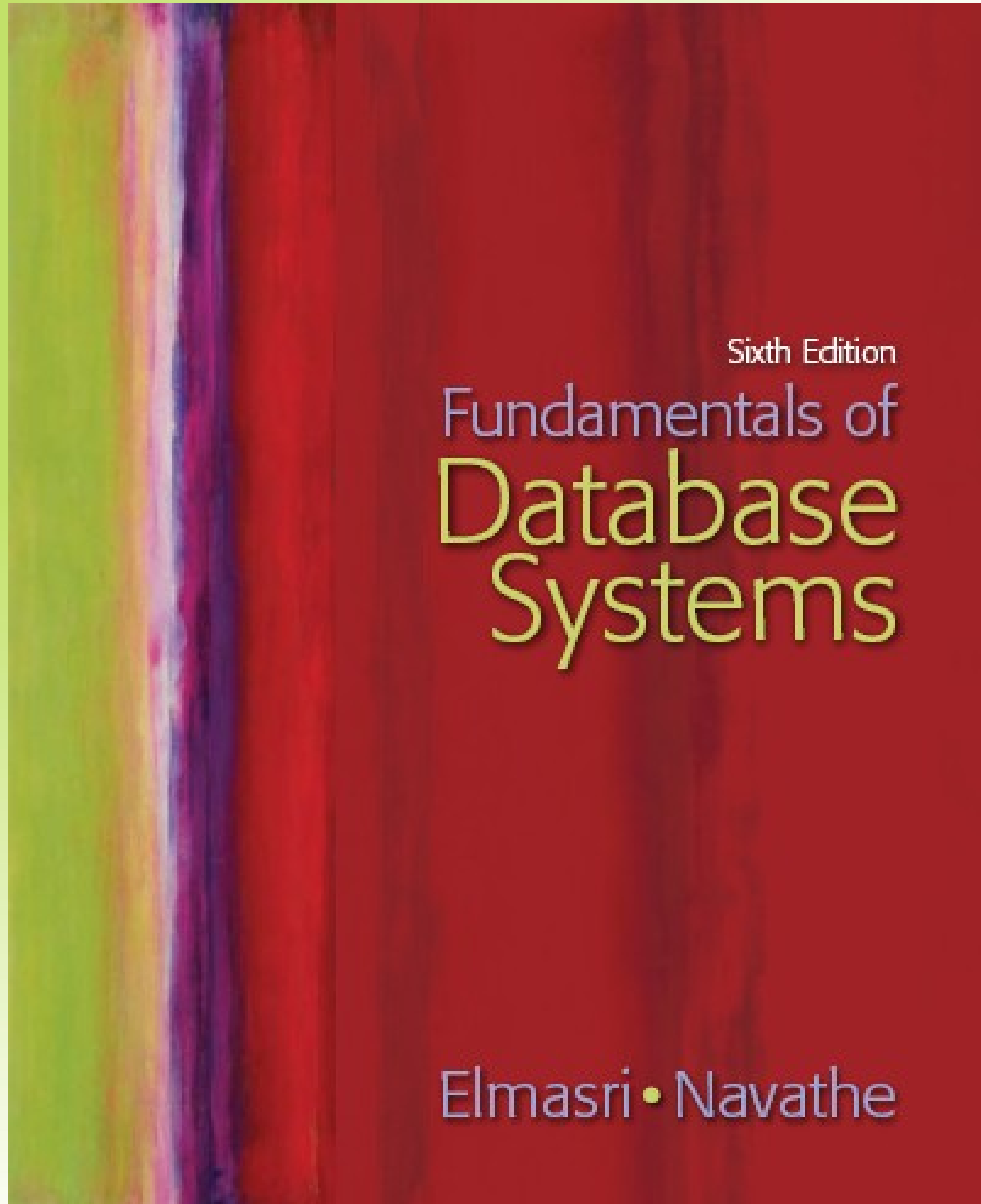
Chapter 22

Concurrency Control Techniques



Addison-Wesley is an imprint of PEARSON

Database Concurrency Control

- 1 Purpose of Concurrency Control
  - To enforce Isolation (through mutual exclusion) among conflicting transactions.
  - To preserve database consistency through consistency preserving execution of transactions.
  - To resolve read-write and write-write conflicts.

- Example:
  - In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Database Concurrency Control

Two-Phase Locking Techniques

- Locking is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction.
- Example:
  - Lock (X). Data item X is locked in behalf of the requesting transaction.
- Unlocking is an operation which removes these permissions from the data item.
- Example:
  - Unlock (X): Data item X is made available to all other transactions.
- Lock and Unlock are Atomic operations.

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Two locks modes:
  - (a) shared (read)   (b) exclusive (write).
- Shared mode: shared lock (X)
  - More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X)
  - Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.
- Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Lock Manager:
  - Managing locks on data items.
- Lock table:
  - Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Database requires that all transactions should be well-formed. A transaction is well-formed if:
  - It must lock the data item before it reads or writes to it.
  - It must not lock an already locked data items and it must not try to unlock a free data item.

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- The following code performs the lock operation:

```
B: if LOCK (X) = 0 (*item is unlocked*)
   then LOCK (X) ← 1 (*lock the item*)
   else begin
            wait (until lock (X) = 0 and
            the lock manager wakes up the transaction);
   goto B
   end;
```

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- The following code performs the unlock operation:

  LOCK (X) ← 0 (*unlock the item*)

  if any transactions are waiting then

  wake up one of the waiting the transactions;

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- The following code performs the read operation:

```
B: if LOCK (X) = “unlocked” then
 begin LOCK (X) ← “read-locked”;
     no_of_reads (X) ← 1;
 end
 else if LOCK (X) = “read-locked” then
          no_of_reads (X) ← no_of_reads (X) +1
      else begin wait (until LOCK (X) = “unlocked” and
                 the lock manager wakes up the transaction);
                 go to B
               end;
```

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- The following code performs the write lock operation:

```
B: if LOCK(X) = “unlocked”
        then LOCK(X) ← “write-locked”
  else begin wait (until LOCK(X) = “unlocked” and
        the lock manager wakes up the transaction);
        go to B
  end;
```

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- The following code performs the unlock operation:

```
if LOCK (X) = "write-locked" then
begin LOCK (X) ← "unlocked";
     wakes up one of the transactions, if any
end
else if LOCK (X) ← "read-locked" then
    begin
        no_of_reads (X) ← no_of_reads (X) -1
        if  no_of_reads (X) = 0 then
        begin
                LOCK (X) = "unlocked";
               wake up one of the transactions, if any
        end
    end;
```

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Lock conversion
  - Lock upgrade: existing read lock to write lock

    if Ti has a read-lock (X) and Tj has no read-lock (X) ($i \neq j$) then
    convert read-lock (X) to write-lock (X)
    else
    force Ti to wait until Tj unlocks X

  - Lock downgrade: existing write lock to read lock

    Ti has a write-lock (X) (*no transaction can have any lock on X*)
    convert write-lock (X) to read-lock (X)

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

- Two Phases:
  - (a) Locking (Growing)
  - (b) Unlocking (Shrinking).
- **Locking (Growing) Phase:**
  - A transaction applies locks (read or write) on desired data items one at a time.
- **Unlocking (Shrinking) Phase:**
  - A transaction unlocks its locked data items one at a time.
- **Requirement:**
  - For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

T1	**T2**	**Result**
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2
write_lock (X);	write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

T1	T2
read_lock (Y); read_item (Y); **unlock (Y);**	
	read_lock (X); read_item (X); **unlock (X);** **write_lock (Y);** read_item (Y); Y:=X+Y; write_item (Y); unlock (Y);
write_lock (X); read_item (X); X:=X+Y; write_item (X); unlock (X);	

Time

Result

X=50; Y=50
Nonserializable because it.
violated two-phase policy.

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

<u>**T’1**</u>	<u>**T’2**</u>	
read_lock (Y);	read_lock (X);	T1 and T2 follow two-phase policy but they are subject to deadlock, which must be dealt with.
read_item (Y);	read_item (X);	
write_lock (X);	Write_lock (Y);	
unlock (Y);	unlock (X);	
read_item (X);	read_item (Y);	
X:=X+Y;	Y:=X+Y;	
write_item (X);	write_item (Y);	
unlock (X);	unlock (Y);	

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

- Two-phase policy generates two locking algorithms
  - (a) **Basic**
  - (b) **Conservative**
- **Conservative**:
  - Prevents deadlock by locking all desired data items before transaction begins execution.
- **Basic**:
  - Transaction locks data items incrementally. This may cause deadlock which is dealt with.
- **Strict**:
  - A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock**

<u>**T'1**</u>	<u>**T'2**</u>
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (X);
write_lock (X);	
(waits for X)	write_lock (Y);
	(waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

- Deadlock (T'1 and T'2)

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock prevention**
  - A transaction locks all data items it refers to before it begins execution.
  - This way of locking prevents deadlock since a transaction never waits for a data item.
  - The conservative two-phase locking uses this approach.

Database Concurrency Control

Dealing with Deadlock and Starvation

Deadlock detection and resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.

To construct and maintain a wait-for graph.

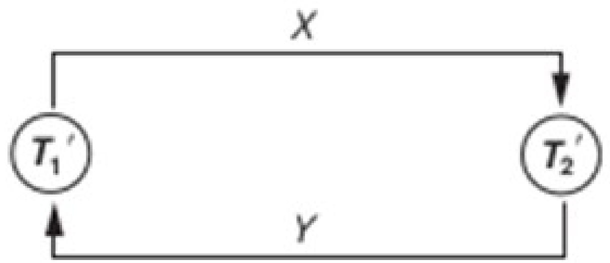
- One node is created in the wait-for graph for each transaction that is currently executing.
- Whenever a transaction Ti is waiting to lock an item X that is currently locked by a transaction Tj, a directed edge (Ti → Tj) is created in the wait-for graph.
- When Tj releases the lock(s) on the items that Ti was waiting for, the directed edge is dropped from the wait-for graph.
- We have a state of deadlock if and only if the wait-for graph has a cycle.

(a)

Time ↓

T_1'	T_2'
read_lock(Y);	
read_item(Y);	
	read_lock(X);
	read_item(X);
write_lock(X);	
	write_lock(Y);

(b)



Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock avoidance**
  - There are many variations of two-phase locking algorithm.
  - Some avoid deadlock by not letting the cycle to complete.
  - That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
  - Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.

Database Concurrency Control

Dealing with Deadlock and Starvation

Deadlock avoidance

- Suppose that transaction Ti tries to lock an item X but is not able to because X is locked by some other transaction Tj with a conflicting lock.
- Wait-die. If TS(Ti) < TS(Tj), then (Ti older than Tj) Ti is allowed to wait; otherwise (Ti younger than Tj) abort Ti (Ti dies) and restart it later with the same timestamp.
- Wound-wait. If TS(Ti) < TS(Tj), then (Ti older than Tj) abort Tj (Ti wounds Tj) and restart it later with the same timestamp; otherwise (Ti younger than Tj) Ti is allowed to wait.

Database Concurrency Control

Dealing with Deadlock and Starvation

Starvation

Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.

Solutions for starvation:

- To have a fair waiting scheme, such as using a first-come-first-served queue.
- To increase the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.
- To use higher priorities for transactions that have been aborted multiple times.
- The Wait-Die and Wound-Wait schemes avoid starvation, because they restart a transaction that has been aborted with its same original timestamp.

Database Concurrency Control

Timestamp based concurrency control algorithm

- **Timestamp**
  - A monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation.
  - Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

Database Concurrency Control

Timestamp based concurrency control algorithm

- **Basic Timestamp Ordering**
  - 1. Transaction T issues a write_item(X) operation:
    - If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then an younger transaction has already read the data item so abort and roll-back T and reject the operation.
    - If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).
  - 2. Transaction T issues a read_item(X) operation:
    - If write_TS(X) > TS(T), then an younger transaction has already written to the data item so abort and roll-back T and reject the operation.
    - If write_TS(X) $\leq$ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Database Concurrency Control

Timestamp based concurrency control algorithm

- **Strict Timestamp Ordering**
  - 1. Transaction T issues a write_item(X) operation:
    - If TS(T) > read_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).
  - 2. Transaction T issues a read_item(X) operation:
    - If TS(T) > write_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

Database Concurrency Control

Timestamp based concurrency control algorithm

- **Thomas's Write Rule**
  - If read_TS(X) > TS(T) then abort and roll-back T and reject the operation.
  - If write_TS(X) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
  - If the conditions given in 1 and 2 above do not occur, then execute write_item(X) of T and set write_TS(X) to TS(T).