CONCURRENCY CONTROL TECHNIQUES

Dr. Dang Tran Khanh
Dept. of Information Systems, CSE/HCMUT
khanh@cse.hcmut.edu.vn

Outline

- Purposes of databases concurrency control
- Two-Phase Locking
- Dealing with deadlock & starvation
- Concurrency control in indexes
- Exercises
- Summary

Purposes of Databases 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 A and if the other transaction is rolled-back or waits

Two-Phase Locking Techniques

- Locking is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction
- Unlocking is an operation which removes read/write permissions from the data item
- Lock and Unlock are Atomic operations (implemented as indivisible units, critical sections in OS - http://en.wikipedia.org/wiki/Critical_sections)

Two-Phase Locking Techniques

- Essential components
  - Two locks modes: shared (read), exclusive (write)
  - Shared mode: Shared lock (X)
    - More than one transaction can apply a shared lock on X for reading but no write lock can be applied on X by any other ones
  - 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

Two-Phase Locking Techniques

- Essential components
  - Lock Manager: managing locks on data items
  - Lock table: Lock manager uses it to store the ID 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

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

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

Two-Phase Locking Techniques

- Essential components
  - 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;

Two-Phase Locking Techniques

- Essential components

  - 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;
```

Two-Phase Locking Techniques

- 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;
```

Two-Phase Locking Techniques

- Write 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;

Two-Phase Locking Techniques

- Lock conversion:
  - Lock upgrade: from 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: from 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)

Two-Phase Locking Techniques

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

Two-Phase Locking Techniques

- Serializability is not guaranteed without Two-Phase Locking protocol

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);	

Two-Phase Locking Techniques

- Problem:

T1	T2	Result
read_lock (Y); read_item (Y); **unlock (Y);**		X=50; Y=50 Nonserializable because it violated the two-phase locking protocol
	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

Two-Phase Locking Techniques

- 2PL:
  - All locking operations (read_lock, write_lock) precede the ***first*** unlock operation in the transaction

Two-Phase Locking Techniques

- With Two-Phase Locking Techniques

T'1

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

T'2

read_lock (X);
read_item (X);
Write_lock (Y);
unlock (X);
read_item (Y);
Y:=X+Y;
write_item (Y);
unlock (Y);

T1 and T2 follow two-phase policy but they are subject to deadlock, which must be dealt with

Two-Phase Locking Techniques

- **Two-phase policy generates two locking algorithms: Basic & 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 must be dealt with
  - **Strict**: Unlocking Write locks is performed *after* a transaction terminates (commits or aborts and rolled-back)
    - This is the most commonly used two-phase locking algorithm
    - Strict 2PL is ***not*** deadlock-free
  - **Rigorous 2PL**: Strict 2PL + Read locks (both Read & Write locks are of concern)

Dealing with Deadlock and Starvation

T'1	**T'2**
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (Y);
write_lock (X);	
(waits for X)	write_lock (Y);
	(waits for Y)

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

Deadlock (T'1 and T'2) here

Dealing with Deadlock and Starvation

- Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further
- Homework: read sections 18.1.3, 18.2 [1] for the details of existing solutions

Concurrency control in indexes

- Problem: as a write_lock is required, the tree must be locked from the root *exclusively*
- Read section 18.6 (and section 18.7 for more information)
- More reading:
  - P.A. Bernstein & E. Newcomer: *Principles of Transaction Processing*, Morgan Kaufmann Publisher, Inc., 2nd Ed., 2009

Exercises

- In the class: Problem 18.20 (p. 653)
  - Prove that the basic 2PL protocol guarantees conflict serializability of schedules
    - Hint: Show that, if a serializability graph for a schedule has a cycle, then at least one of the transactions participating in the schedule does not obey the 2PL protocol

- Homework: Chapter 19 (self-reading)

SUMMARY

- Purposes of Databases Concurrency Control
- Two-Phase Locking
- Dealing with deadlock & starvation
- Do not forget: Homework !!
- Next lecture: Introduction to DB Security