Database Management Systems

Dr. Santhosh Manikonda
Department of CSE
School of Engineering

Malla Reddy University, Hyderabad

Syllabus
UNIT - V

- Concurrency Control Techniques:
  - Two-Phase Locking Techniques for Concurrency Control
  - Concurrency Control Based on Timestamp Ordering.

- Disk Storage, Basic File Structures:
  - Introduction
  - Secondary Storage Devices
  - Buffering of Blocks
  - Placing File Records on Disk
  - Operations on Files.

Concurrency Control Protocols

- Primary goal of concurrency protocols is to achieve consistency
- This goal is achieved by using different protocols
  - Shared/Exclusive Lock
  - Two – Phase Locking

Shared/Exclusive Lock

- Shared Lock (S): In shared lock, a transaction is allowed to only read
  - Shared lock allows another transaction(s) for shared lock only
- Exclusive Lock (X): In exclusive lock, a transaction is allowed to read and write
  - Exclusive lock does not allow any lock until it is released
  - If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Shared/Exclusive Lock

		Request	
		S	X
Grant	S	**YES**	**NO**
	X	**NO**	**NO**

Shared/Exclusive Lock

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
X(B)	
R(B)	
W(B)	
U(B)	

Shared/Exclusive Lock

- Problems in Shared Lock/Exclusive Lock
  - May not be sufficient to achieve serializable schedule
  - May not be recoverable
  - May not be free from deadlock
  - May not be free from starvation

Shared/Exclusive Lock

- May not be recoverable

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)
	W(A)
	U(A)
	Commit
X(B)	
R(B)	
W(B)	
U(B)	
Failed	

Shared/Exclusive Lock

- May not be recoverable

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)
	W(A)
	U(A)
	Commit
X(B)	
R(B)	
W(B)	
U(B)	
Failed	

Shared/Exclusive Lock

- May not be free from deadlock

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

Wait

Wait

Shared/Exclusive Lock

- May not be free from starvation

T1	T2	T3	T4
S(A)			
	X(A)		
		S(A)	
U(A)			
			S(A)
		U(A)	

Two – Phase Lock (2PL)

- Growing Phase:
  - In this phase a transaction may acquire locks
  - Transaction may not release locks

- Shrinking Phase:
  - In this phase Transaction may release locks
  - Transaction may not obtain locks

- The protocol assures serializability.
- It can be proved that the transactions can be serialized in the order of their lock points (i.e., the point where a transaction acquired its final lock).

Two – Phase Lock (2PL) – Lock Conversions

- Two-phase locking with lock conversions:
- – First Phase:
  - can acquire a lock-S on item
  - can acquire a lock-X on item
  - can convert a lock-S to a lock-X (upgrade)
- – Second Phase:
  - can release a lock-S
  - can release a lock-X
  - can convert a lock-X to a lock-S (downgrade)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Two – Phase Lock (2PL)

Growing Phase

T1	T2
X(A)	
R(A)	
W(A)	
	X(A)
	R(A)
	W(A)
X(B)	
R(B)	
W(B)	
$\boxed{\text{U(A)}}$	
U(B)	

Shrinking Phase

Two – Phase Lock (2PL)

T1	T2
S(A)	
R(A)	
X(B)	S(A)
R(B)	R(A)
W(B)	W(A)
.	S(D)
.	R(D)
.	.
U(A)	.
U(B)	.

Automatic Acquisition of Locks

- A transaction Ti issues the standard read/write instruction, without explicit locking calls.
- The operation read(D) is processed as:

```
 if Ti has a lock on D
      then
     read(D)
else
      begin
           if necessary wait until no other
           transaction has a lock-X on D
         grant Ti a  lock-S on D;
         read(D)
end
```

Automatic Acquisition of Locks

- The operation write(D) is processed as:

```
if Ti has a  lock-X on D
      then
       write(D)
    else begin
        if necessary wait until no other transaction
      has any lock on D,
        if Ti has a lock-S on D
             then
                upgrade lock on D  to lock-X
            else
                grant Ti a lock-X on D
            write(D)
      end;
```

- All locks are released after commit or abort

Two – Phase Lock (2PL)

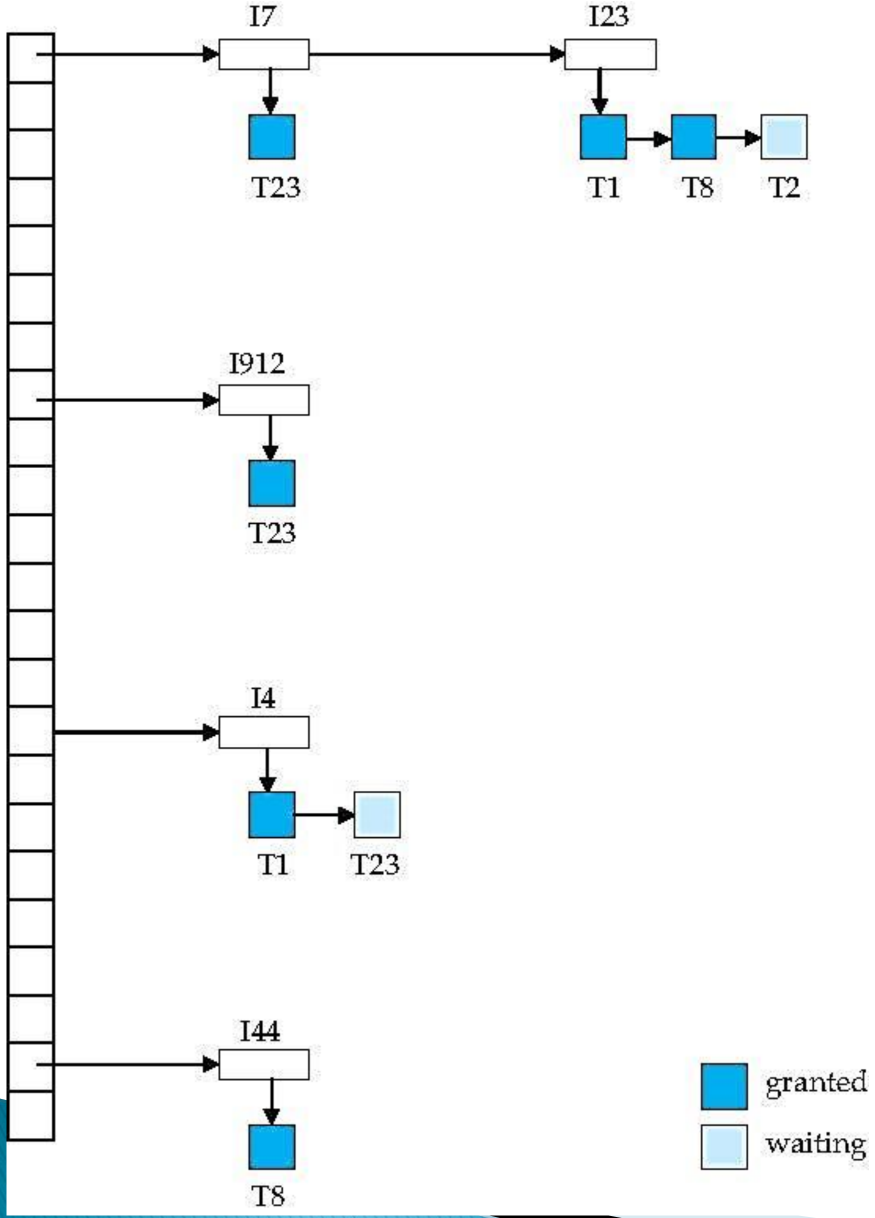
- Strict 2PL
  - Strict 2PL should satisfy basic 2PL
  - All Exclusive Locks should be held until Commit/Abort

- Rigorous 2PL:
  - Rigorous 2PL should satisfy basic 2PL
  - All Exclusive Locks, Shared Locks should be held until Commit/Abort

Implementation of Locking

- A lock manager can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a lock table to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked

Lock Table



- Dark blue rectangles indicate granted locks;
- Light blue indicate waiting requests
- Lock table also records the type of lock granted or requested
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted
- If transaction aborts, all waiting or granted requests of the transaction are deleted
- Lock manager may keep a list of locks held by each transaction, to implement this efficiently

Timestamp-Based Protocol

- Each transaction is issued a timestamp when it enters the system
  - It is a unique value assigned to every transaction
  - It tells the order in which a transaction has entered the system
    - Transaction: T_i
    - Time stamp: TS(T_i)
- If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that $TS(T_i) < TS(T_j)$
- The protocol manages concurrent execution such that the time-stamps determine the serializability order – *Older transactions are executed first*

Timestamp-Based Protocol

- In order to assure such behavior, the protocol maintains for each data A two timestamp values:
- W-timestamp(A)
  - WTS(A) is the largest time-stamp of any transaction that executed write(A) successfully – Last transaction which performed Write successfully
- R-timestamp(A)
  - RTS(A) is the largest time-stamp of any transaction that executed read(A) successfully – Last transaction which performed Read successfully

Timestamp-Based Protocol

- **Time Stamp of Transaction $TS(T_i)$**

10:00	**10:05**	**10:07**	**(Time of Transaction)**
T1	T2	T3	**Ti**
100	120	134	**$TS(T_i)$**
Oldest		*Youngest*	

Timestamp-Based Protocol

- **Time stamp of Data Item RTS(A)**

09:00	**09:03**	**09:15**	
T1	T2	T3	**Ti**
10	12	24	**TS(T_i)**
R(A)	R(A)	***R(A)***	

RTS(A) = 24

Timestamp-Based Protocol

- **Time stamp of Data Item WTS(A)**

09:00	**09:03**	**09:15**	
T1	T2	T3	**Ti**
10	12	24	**TS(T_i)**
W(A)	**W(A)**	*W(A)*	

WTS(A) = 12

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Read(A)
- If TS(T_i) <WTS(A), then T_i needs to read a value of A that was already overwritten.
  - Hence, the read operation is rejected, and T_i is rolled back
- If TS(T_i) ≥ WTS(A), then then the read operation is executed, and set RTS(A) = max{RTS(A), TS(T_i)}

Example
TS(T_i) <WTS(A)
900 < 903

09:00	09:03
Ti	Tx
	W(A)
	.
R(A)	.
	.
	xxxx

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Read(A)
- If TS(T_i) <WTS(A), then T_i needs to read a value of A that was already overwritten.
  - Hence, the read operation is rejected, and T_i is rolled back
- If TS(T_i) ≥ WTS(A), then the read operation is executed, and set
  RTS(A) = max{RTS(A), TS(T_i)}

Example
TS(T_i) >WTS(A)
910 > 903

09:10	**09:03**
Ti	Tx
	W(A)
R(A)	

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Write(A)
- If TS(T_i) $<$ RTS(A), then the value of A that T_i is producing was needed previously, and the system assumed that that value would never be produced.
  - Hence, the write operation is rejected, and T_i is rolled back

- If TS(T_i) $<$ WTS(A), then T_i is attempting to write an obsolete value of A
  - Hence, the write operation is rejected, and T_i is rolled back

- Otherwise, the write operation is executed, and set WTS(A) = max{WTS(A), TS(T_i)}

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Write(A)
- If TS(T_i) $<$ RTS(A), then the value of A that T_i is producing was needed previously, and the system assumed that that value would never be produced.
  - Hence, the write operation is rejected, and T_i is rolled back
- If TS(T_i) $<$ WTS(A), then T_i is attempting to write an obsolete value of A
  - Hence, the write operation is rejected, and T_i is rolled back
- Otherwise, the write operation is executed, and set WTS(A) = max{WTS(A), TS(T_i)}

Example
TS(T_i) $<$RTS(A)
900 $<$ 903

09:00	09:03
Ti	Tx
	R(A)
W(A)	

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Write(A)
- If TS(T_i) ≤ RTS(A), then the value of A that T_i is producing was needed previously, and the system assumed that that value would never be produced.
  - Hence, the write operation is rejected, and T_i is rolled back
- If TS(T_i) < WTS(A), then T_i is attempting to write an obsolete value of A
  - Hence, the write operation is rejected, and T_i is rolled back
- Otherwise, the write operation is executed, and set WTS(A) = max{WTS(A), TS(T_i)}

Example
TS(T_i) <WTS(A)
900 < 903

09:00	09:03
Ti	Tx
	W(A)
W(A)	

Timestamp-Based Protocol

- Suppose a transaction T_i issues a Write(A)
- If TS(T_i) ≤ RTS(A), then the value of A that T_i is producing was needed previously, and the system assumed that that value would never be produced.
  - Hence, the write operation is rejected, and T_i is rolled back
- If TS(T_i) < WTS(A), then T_i is attempting to write an obsolete value of A
  - Hence, the write operation is rejected, and T_i is rolled back
- Otherwise, the write operation is executed, and set WTS(A) = max{WTS(A), TS(T_i)}

Example TS(T_i) > RTS(A)	Example TS(T_i) > WTS(A)

Timestamp-Based Protocol - Properties

- It ensures conflict serializability
- It ensures view serializability
- Free from deadlock
- Possibility of dirty read and irrecoverable schedule