Database Management Systems

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Lecture Session-17

Concurrency Control

Contents

- *Introduction to Concurrency Control*
- *Implementing Serializability*
- *Lock-based protocols*
- *Deadlock condition*
- *Two-phase locking protocol*
- *Time-phase locking protocol*

Introduction

❖ In a DBMS multiple transactions are executed concurrently.

❖ If the transactions are executed concurrently then the resources can be utilized more efficiently hence more throughput is achieved.

❖ Here, for transactions we consider data items as resources because transactions process data by accessing them.

❖ When multiple transactions access data elements in a concurrent way, this may destroy the consistency of the database.

Implementing Serializabilty

One way to ensure *serializability* is to allow the transactions to access the data items in a mutually exclusive manner.

This is to make sure that when one transaction access a data item no other transaction can modify that data item.

The following techniques implement mutual exclusion and control concurrency.

1. *Lock-based protocols*
2. *Timestamp-based protocols*

***1. Concurrency Control Using* Locks**: A data item may be locked in various modes.

i) ***Shared*** (denoted by *S*): if a transaction obtains a shared mode lock on a data item Q, it can read Q but not modify Q.

(ii) ***Exclusive*** (denoted by *X*): if this lock is obtained, a transaction can read or write the data item.

Lock Compatibility Matrix

	S	X
S	True	False
X	False	False

This says that if a transaction T_i obtains a lock on a data item in *S-mode*, other transaction can get a lock on the same item in *S-mode* but not in *X-mode*.

If a transaction obtains a lock in *X-mode* on a data item no other transaction can obtain a lock on the same data item in any mode.

Deadlock

The Mutual exclusion mechanism leads to deadlock situation.

For example, if transaction T_i holds a lock on a data item (Q) in *X-mode* and waits for a lock on another data item (P) which is locked by another transaction T_j in *X-mode*, further to release the lock on *P*, T_j must acquire a lock on Q, which is locked by T_i. This is a circular wait condition and results in a *deadlock* situation.

Wait-for Graph

Deadlock condition can be determined by a *wait-for* graph.

All transactions of the schedule become vertices.

And we have an edge between two transactions T_i and T_j. if T_i is waiting for T_j to release a lock on a data item.
If the graph has a cycle then we can say that the schedule will result in a deadlock.

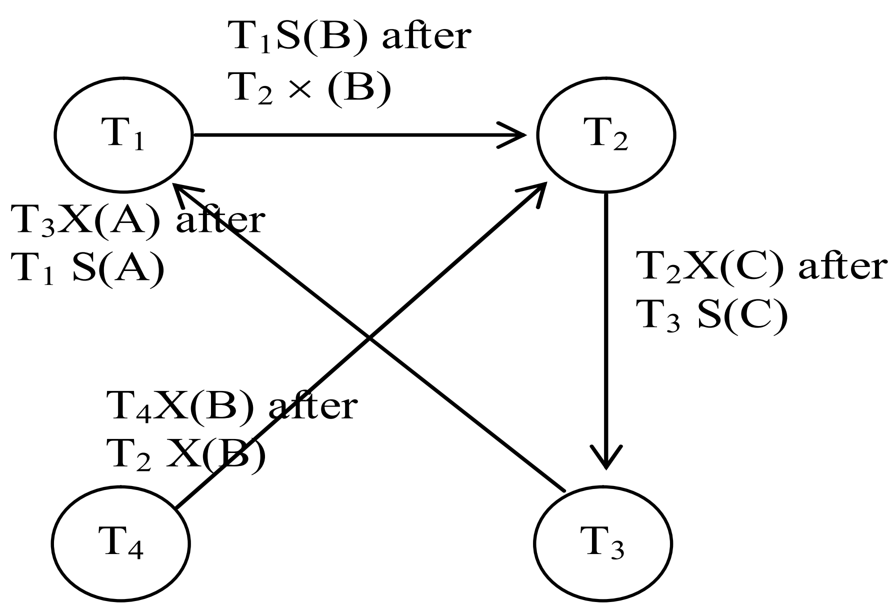
T_1	T_2	T_3	T_4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	

S(A) means transaction locks A in share mode

R(A) – transaction reads A

X(C) – Transaction locks C in X-mode

W(B) – transaction write B



In the above graph there exists a cycle hence this schedule leads to deadlock.
If a transaction T_i requests a lock and transaction T_j holds a conflicting lock. The lock manager can use one of the following *policies to prevent deadlocks*.

Timestamp based:
Wait-Die: If T_i is older than T_j it is allowed to wait otherwise aborted.
Wound-wait: If T_i older than T_j allowed to run by aborting T_j else T_i will wait.

Priority based
Wait-Die: If T_i has higher priority than T_j it is allowed to wait otherwise aborted.
Wound-wait: If T_i lies higher priority it is allowed to run by aborting T_j else T_i will wait.

Two-phase locking protocol:

This protocol answers serializability.

According to this each transaction issues lock and unlock requests in two phases.

i) *Growing phase*: In this phase, a transaction may obtain locks but may not release any lock.

ii) *Shrinking phase*: In this phase, a transaction may release locks but may not obtain any new locks.

The two-phase locking protocol ensures conflict serializability.

It does not ensure freedom from deadlock.

2. *Timestamp-based Concurrency Control*

Maintaining the ordering between every pair of conflicting transactions is significant.

If we select the ordering in advance, we can achieve serializability. Time-stamping is a method to fix the ordering.

Each transaction is assigned a unique fixed timestamp.
If $TS(T_i) < TS(T_j)$, this implies that T_i should be executed before T_j.

The time-stamps determine the serializability order.
Each data item is associated with two timestamp values.

W-timestamp(Q) – represents the largest timestamp of any transaction that successfully executes Write(Q).

R-timestamp(Q) - which denotes the largest time stamp of any transaction that successfully executed Read(Q).

These values are updated whenever read(Q) or write(Q) are executed.

Timestamp ordering Protocol:

This protocol operates as follows:

i)*Suppose Transaction T_i* issues read(Q)

If $TS(T_i)$ < W-stamp(Q), then it implies that T_i need to read Q which was already overwritten.

Hence read operation is rejected and T_i is rolled back.

If $TS(T_i) \geq$ W-timestamp (Q) then read operation is executed.

ii)*Suppose T_i issues write (Q)*

If $TS(T_i)$ < R-timestamp(Q) it implies that the value of Q being produced by T_i had to be written long back.

Hence reject T_i & roll back.

If $TS(T_i)$ < W-timestamp(Q), T_i is attempting to write some absolute value of Q.

Hence reject T_i & roll back.

Otherwise write operation is executed.

Summary

- ✓ *Concepts related to Concurrency Control*
- ✓ *Approaches for Implementing Serializability*
- ✓ *How lock-based protocols work*
- ✓ *Detecting the Deadlock condition and resolving*
- ✓ *Two-phase locking protocol*
- ✓ *How timestamp-based protocol works*