Database Management System 21

Concurrency Control

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Need of Concurrency Control

Lost Update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect

T_1	T_2
Read(A);	
A:=A-100;	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
Write(A);	
	Write(A);
Read(B);	
B:=B+100;	
Write(B);	

Need of Concurrency Control...

Temporary Update(or Dirty Read) Problem

This problem occurs when one transaction updates a database item and then the transaction fails due to some reason. The updated item is accessed by another transaction before it is changed back to its original value

T_1	T_2
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
	Write(A);
Read(B);	
B:=B+100;	

Need of Concurrency Control...

Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated

T_1	T_2
	sum=0;
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	sum:=sum+A;
	Read(B);
	sum:=sum+B;
Read(B);	
B:=B+100;	
Write(B);	

Lock-Based Protocols

Locking is a procedure used to control concurrent access to data. Locks enable a multi-user database system to maintain the integrity of transactions by isolating a transaction from others executing concurrently

Locking is one of the most widely used mechanisms to ensure serializability

Data items can be locked in two modes:

- **Shared lock or Read lock**: If a transaction T_i has obtained a shared mode lock(S) on data item Q, then T_i can only read the data item Q, but cannot write on Q
- **Exclusive lock or Write lock**: If a transaction T_i has obtained an exclusive mode lock(X) on data item Q, then T_i can both read and write Q

A transaction must obtain a lock on a data item before it can perform a read or write operation

Basic Rules for Locking

Basic Rules for Locking

- If a transaction has a read lock on a data item, it can only read the item; but cannot update its value
- If a transaction has a read lock on a data item, other transactions can obtain read locks on the same data item, but they cannot obtain any update lock on it
- If a transaction has a write lock on a data item, then it can both read and update the value of that data item
- If a transaction has a write lock on a data item, then other transactions cannot obtain either a read lock or a write lock on that data item

A transaction requests a shared lock on data item Q by executing the Lock-S(Q) instruction
Similarly, a transaction can request an exclusive lock through the Lock-X(Q) instruction
A transaction can unlock a data item Q by the Unlock(Q) instruction

Working of Locking

Working of Locking

- All transactions that need to access a data item must first acquire a read lock or write lock on the data item depending on whether it is a read only operation or not
- If the data item for which the lock is requested is not already locked, then the transaction is granted with the requested lock immediately
- If the item is currently locked, the database system determines what kind of lock is the current one. Also, it finds out which type of lock is requested:
  - If a read lock is requested on a data item that is already under a read lock, then the request will be granted
  - If a write lock is requested on a data item that is already under a read lock, then the request will be denied
  - Similarly; if a read lock or a write lock is requested on a data item that is already under a write lock, then the request is denied and the transaction must wait until the lock is released

Working of Locking...

Working of Locking...

- A transaction continues to hold the lock until it explicitly releases it either during the execution or when it terminates
- The effects of a write operation will be visible to other transactions only after the lock is released

A concurrent schedule, which is conflict serializable to a serial schedule, will always get the respective locks from the concurrency control manager

But, if the concurrent schedule is not conflict serializable, the requested locks will not be granted by the concurrency control manager

However, in case of Incorrect Summary Problem, all the requested locks will be granted resulting in incorrect values

Working of Locking...

Schedule3

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
	Write(A);
Read(B);	
Write(B);	
	Read(B);
	Write(B);

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
Write(A);		
Unlock(A)		
	Lock-X(A)	
		Grant-X(A, T_2)
	Read(A);	
	Write(A);	
	Unlock(A)	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
Write(B);		
Unlock(B)		
	Lock-X(B)	
		Grant-X(B, T_2)
	Read(B);	
	Write(B);	
	Unlock(B)	

Working of Locking...

Schedule4

T_1	T_2
Read(A);	
	Read(A);
	Write(A);
	Read(B);
Write(A);	
Read(B);	
Write(B);	
	Write(B);

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
	Lock-X(A)	

Working of Locking...

Schedule5

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
Read(B);	
	Write(A);
	Read(B);
	Write(B);
Write(B);	

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
Write(A);		
Unlock(A)		
	Lock-X(A)	
		Grant-X(A, T_2)
	Read(A);	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
	Write(A);	
	Unlock(A)	
	Lock-X(B)	

Working of Locking...

T_1	T_2
Read(A);	Read(A);
A:=A-100;	Read(B);
Write(A);	Display(A+B);
Read(B);	
B:=B+100;	
Write(B);	

T_1	T_2
Lock-X(A);	Lock-S(A);
Read(A);	Read(A);
A:=A-100;	Unlock(A);
Write(A);	Lock-S(B);
Unlock(A);	Read(B);
Lock-X(B);	Unlock(B);
Read(B);	Display(A+B);
B:=B+100;	
Write(B);	
Unlock(B);	

Working of Locking...

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
A:=A-100;		
Write(A);		
Unlock(A)		
	Lock-S(A)	
		Grant-S(A, T_2)
	Read(A);	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B, T_2)
	Read(B);	
	Unlock(B)	
	Display(A+B);	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
B:=B+100;		
Write(B);		
Unlock(B)		

Though the concurrency control manager will not face any problem in granting the locks, the above schedule gives incorrect result for transaction T_2

Working of Locking...

To solve the previous discussed problem, different alternative solutions are possible. One solution can be by delaying the unlocking process. That means the unlocking is delayed to the end of the transaction

Unfortunately, this type of locking can lead to an undesirable situation

T_1	T_2
Lock-X(A);	Lock-S(A);
Read(A);	Read(A);
A:=A-100;	Lock-S(B);
Write(A);	Read(B);
Lock-X(B);	Display(A+B);
Read(B);	Unlock(A);
B:=B+100;	Unlock(B);
Write(B);	
Unlock(A);	
Unlock(B);	

Working of Locking...

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		Grant-X(A, T_1)
Read(A);		
A:=A-100;		
Write(A);		
	Lock-S(A)	

Since T_1 is holding an exclusive-lock on A and T_2 is requesting a shared-lock on A, the concurrency control manager will not grant the lock permission to T_2. Thus, T_2 is waiting for T_1 to unlock A

T_3
Lock-X(B);
Read(B);
B:=B-100;
Write(B);
Lock-X(A);
Read(A);
A:=A+100;
Write(A);
Unlock(B);
Unlock(A);

Working of Locking...

T_3	T_2	Concurrency-Control Manager
Lock-X(B)		
		Grant-X(B, T_3)
Read(B);		
B:=B-100;		
Write(B);		
	Lock-S(A)	
		Grant-S(A, T_2)
	Read(A);	
	Lock-S(B);	
Lock-X(A)		

T_2 is waiting for T_3 to unlock B. Similarly, T_3 is waiting for T_2 to unlock A. Thus, this is a situation where neither of these transactions can ever proceed with its normal execution. This type of situation is called deadlock

If we do not use locking, or if we unlock data items as soon as possible after reading or writing them, we may get inconsistent states

On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur

Locking Protocol

Locking Protocol

When a transaction requests a lock on a data item in a particular mode, and no other transaction has put a lock on the same data item in a conflicting mode, then the lock can be granted by the concurrency control manager

However, we must take some precautionary measures to avoid the following scenarios:

- Suppose a transaction T_1 has a shared-mode lock on a data item, and another transaction T_2 requests an exclusive-mode lock on that same data item. In this situation, T_2 has to wait for T_1 to release the shared-mode lock
- Suppose, another transaction T_3 requests a shared-mode lock on the same data item while T_1 is holding a shared lock on it. As the lock request is compatible with lock granted to T_1, so T_3 may be granted the shared-mode lock. But, T_2 has to wait for the release of the lock from that data item

Locking Protocol...

Locking Protocol

- At this point, T_1 may release the lock, but still T_2 has to wait for T_3 to finish. There may be a new transaction T_4 that requests a shared-mode lock on the same data item, and is granted the lock before T_3 releases it
- In such a situation, T_2 never gets the exclusive-mode lock on the data item. Thus, T_2 cannot progress at all and is said to be starved. This problem is called as the starvation problem

We can avoid starvation of transactions by granting locks in the following manner; when a transaction T_i requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that:

- There is no other transaction holding a lock on Q in a mode that conflicts with M
- There is no other transaction that is waiting for a lock on Q and that made its lock request before T_i