Database Management System (DBMS)

Lecture-42

Dharmendra Kumar

December 28, 2020

Concurrency Control

Concurrency Control

When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved. To ensure that it is, the system must control the interaction among the concurrent transactions. The mechanism used to control the interaction of transactions is called concurrency control scheme.

There are number of concurrency control schemes.

1. Lock based protocol
2. Time stamp based protocol
3. Validation based protocol
4. Multiple granularity protocol
5. Multi-version protocol

Lock based protocol

A lock is a mechanism to control concurrent access to a data item. Data item can be locked in two modes.

Shared mode(S): If a transaction T_i has obtained a shared-mode lock on item Q, then T_i can read, but cannot write, Q.

Exclusive mode(S): If a transaction T_i has obtained an exclusive-mode lock on item Q, then T_i can both read and write Q.

Note: The transaction makes the lock request to the concurrency control manager. Transaction can process only after lock request is granted.

Concurrency Control

Compatibility function

Given a set of lock modes, we can define a compatibility function on them as follows.
Let A and B represent arbitrary lock modes. Suppose that a transaction T_i requests a lock of mode A on item Q on which transaction T_j ($T_i \neq T_j$) currently holds a lock of mode B. If transaction T_i can be granted a lock on Q immediately, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a function can be represented conveniently by a matrix. An element comp(A, B) of the matrix has the value **true** if and only if mode A is compatible with mode B.

Compatibility Matrix

	S	X
S	true	false
X	false	false

Concurrency Control

Note:

- A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction.
- A transaction requests an exclusive lock on data item Q by executing the lock-X(Q) instruction.
- To unlock the data item Q, we use unlock(Q) instruction.

Note:

To access a data item, transaction T_i must first lock that item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. Thus, T_i is made to wait until all incompatible locks held by other transactions have been released.

Concurrency Control

Example: Consider the following two transactions T_1 and T_2 with locking modes.

Transaction T_1 and T_2

T_1: lock-X(B);
read(B);
$B := B - 50$;
write(B);
unlock(B);
lock-X(A);
read(A);
$A := A + 50$;
write(A);
unlock(A).

T_2: lock-S(A);
read(A);
unlock(A);
lock-S(B);
read(B);
unlock(B);
display($A + B$).

Concurrency Control

Consider the following schedule-1 of these transactions.

Schedule-1

T_1	T_2	concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B)		
$B := B - 50$		
write(B)		
unlock(B)		
	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(A)		
		grant-X(A, T_2)
read(A)		
$A := A + 50$		
write(A)		
unlock(A)		

Concurrency Control

Suppose that the values of accounts A and B are \$100 and \$200, respectively. If these two transactions are executed serially, either in the order T_1, T_2 or the order T_2, T_1, then transaction T_2 will display the value \$300. If, however, these transactions are executed concurrently, then schedule 1 is possible. In this case, transaction T_2 displays \$250, which is incorrect. The reason for this mistake is that the transaction T_1 unlocked data item B too early, as a result of which T_2 saw an inconsistent state.

Concurrency Control

Example: Consider the following two transactions T_3 and T_4 with locking modes.

Transaction T_3 and T_4

T_3: lock-X(B);
read(B);
$B := B - 50$;
write(B);
lock-X(A);
read(A);
$A := A + 50$;
write(A);
unlock(B);
unlock(A).

T_4: lock-S(A);
read(A);
lock-S(B);
read(B);
display($A + B$);
unlock(A);
unlock(B).

Concurrency Control

Consider the following schedule-2 of these transactions.

Schedule-2

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Consider the partial schedule-2 for T_3 and T_4. Since T_3 is holding an exclusive-mode lock on B and T_4 is requesting a shared-mode lock on B, T_4 is waiting for T_3 to unlock B. Similarly, since T_4 is holding a shared-mode lock on A and T_3 is requesting an exclusive-mode lock on A, T_3 is waiting for T_4 to unlock A. Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This situation is called **deadlock**.

Note: When deadlock occurs, the system must roll back one of the two transactions.

Locking protocol This is the set of rules indicating when a transaction may lock and unlock each of the data items.

Note: A schedule S is legal under a given locking protocol if S is a possible schedule for a set of transactions that follow the rules of the locking protocol.

Note: A locking protocol ensures conflict serializability if and only if all legal schedules are conflict serializable.

Starvation

Suppose a transaction T_2 has a shared-mode lock on a data item, and another transaction T_1 requests an exclusive-mode lock on the data item. Clearly, T_1 has to wait for T_2 to release the shared-mode lock. Meanwhile, a transaction T_3 may request a shared-mode lock on the same data item. The lock request is compatible with the lock granted to T_2, so T_3 may be granted the shared-mode lock. At this point T_2 may release the lock, but still T_1 has to wait for T_3 to finish. But again, 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 fact, it is possible that there is a sequence of transactions that each requests a shared-mode lock on the data item, and each transaction releases the lock a short while after it is granted, but T_1 never gets the exclusive-mode lock on the data item. The transaction T_1 may never make progress, and is said to be starved. This situation is said to be **starvation**.

Two-phase locking protocol

This protocol requires that each transaction issue lock and unlock requests in two phases:

1. Growing phase: A transaction may obtain locks, but may not release any lock.

2. Shrinking phase: A transaction may release locks, but may not obtain any new locks.

Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

Concurrency Control

Example: Transactions T_3 and T_4 are locked in two phase. While, transactions T_1 and T_2 are not locked in two phase.

Note: Two-phase locking protocol ensures conflict serializability. The serializability order of transactions will be based on lock point in the transactions.

Lock point: Lock point of a transaction is a point in the schedule where the transaction has obtained its final lock (the end of its growing phase).

Note: Two-phase locking does not ensure freedom from deadlock. Observe that transactions T_3 and T_4 are in two phase, but, in schedule 2, they are deadlocked.

Note: In addition to being serializable, schedules should be cascadeless. Cascading rollback may occur under two-phase locking.

Example: Consider the partial schedule in the following figure:-

Partial schedule under two-phase locking protocol

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

Strict two-phase locking protocol

This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits.

Rigorous two-phase locking protocol

Another variant of two-phase locking is the rigorous two-phase locking protocol, which requires that all locks be held until the transaction commits.

Note: With rigorous two-phase locking, transactions can be serialized in the order in which they commit.

Lock Conversion

Upgrade: We denote conversion from shared to exclusive modes by upgrade.

Downgrade: We denote conversion from exclusive to shared by downgrade.

Note: Lock conversion cannot be allowed arbitrarily. Rather, upgrading can take place in only the growing phase, whereas downgrading can take place in only the shrinking phase.

Note: Strict two-phase locking and rigorous two-phase locking (with lock conversions) are used extensively in commercial database systems.

Note: A simple but widely used scheme automatically generates the appropriate lock and unlock instructions for a transaction, on the basis of read and write requests from the transaction:

- When a transaction T_i issues a read(Q) operation, the system issues a lock- S(Q) instruction followed by the read(Q) instruction.
- When T_i issues a write(Q) operation, the system checks to see whether T_i already holds a shared lock on Q. If it does, then the system issues an upgrade(Q) instruction, followed by the write(Q) instruction. Otherwise, the system issues a lock-X(Q) instruction, followed by the write(Q) instruction.
- All locks obtained by a transaction are unlocked after that transaction commits or aborts.