Database Management Systems (CSN-351)
Concurrency Control

BTech 3rd Year (CS) + Minor + Audit

Instructor: **R**anita **B**iswas
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand - 247 667, India

Lock-Based Protocols

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

Lock-Based Protocols

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

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

Lock-compatibility Matrix

	S	X
S	true	false
X	false	false

Transactions with Locks

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($A + B$).

Schedule 1

T_1	T_2	concurreny-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_1)
read(A)		
$A := A - 50$		
write(A)		
unlock(A)		

Transactions with Delayed Locks

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).

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)	

Problems

Deadlock

Problems

Deadlock

Starvation

Granting of Locks

- There is no other transaction holding a lock on Q in a mode that conflicts with M.

Granting of Locks

- 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.

Two-Phase Locking Protocol

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

Two-Phase Locking Protocol

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

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

Partial Schedule under 2-Phase Locking

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)

Transactions

T_8: read(a_1);
read(a_2);
...
read(a_n);
write(a_1).

T_9: read(a_1);
read(a_2);
display($a_1 + a_2$).

Incomplete Schedule with a Lock Conversion

T_8	T_9
lock-S(a_1)	
	lock-S(a_1)
lock-S(a_2)	
	lock-S(a_2)
lock-S(a_3)	
lock-S(a_4)	
	unlock(a_1)
	unlock(a_2)
lock-S(a_n)	
upgrade(a_1)	

Question 1

Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects $\{O_1, ..., O_k\}$. This is done in the following manner:
Step 1: T acquires exclusive locks to $O_1, ..., O_k$ in increasing order of their addresses.
Step 2: The required operations are performed.
Step 3. All locks are released.
This protocol will

- ◦ guarantee serializability and deadlock-freedom
- ◦ guarantee neither serializability nor deadlock-freedom
- ◦ guarantee serializability but not deadlock-freedom
- ◦ guarantee deadlock-freedom but not serializability

Locking Protocols

Two-phase locking → Serializable schedules

Locking Protocols

Two-phase locking $\rightarrow$ Serializable schedules

Strict two-phase locking $\rightarrow$ Cascadeless schedules

Locking Protocols

Two-phase locking → Serializable schedules

Strict two-phase locking → Cascadeless schedules

Rigorous two-phase locking → Strict schedules

Timestamp-Based Protocols

Transaction timestamps: With each transaction T_i in the system, we associate a unique fixed timestamp, denoted by $TS(T_i)$.

Timestamp-Based Protocols

Transaction timestamps: With each transaction T_i in the system, we associate a unique fixed timestamp, denoted by $TS(T_i)$.

Data item timestamps $\rightarrow$ with each data item Q two timestamp values are associated.

- **W-timestamp(Q)** denotes the largest timestamp of any transaction that executed $write(Q)$ successfully.

- **R-timestamp(Q)** denotes the largest timestamp of any transaction that executed $read(Q)$ successfully.

Timestamp-Ordering Protocol

T_i issues $read(Q)$

Timestamp-Ordering Protocol

T_i issues $read(Q)$

(a) If $TS(T_i) <$ W-timestamp(Q):

(b) If $TS(T_i) \geqslant$ W-timestamp(Q):

Timestamp-Ordering Protocol

T_i issues $read(Q)$

(a) If $TS(T_i) <$ W-timestamp(Q):

(b) If $TS(T_i) \geqslant$ W-timestamp(Q):

T_i issues $write(Q)$

Timestamp-Ordering Protocol

T_i issues $read(Q)$

(a) If $TS(T_i) <$ W-timestamp(Q):

(b) If $TS(T_i) \geqslant$ W-timestamp(Q):

T_i issues $write(Q)$

(a) If $TS(T_i) <$ R-timestamp(Q):

(b) If $TS(T_i) <$ W-timestamp(Q):

(c) Else:

Example Schedule

T_{25}	T_{26}
read(B)	
	read(B)
	$B := B - 50$
	write(B)
read(A)	
	read(A)
display($A + B$)	
	$A := A + 50$
	write(A)
	display($A + B$)

Thomas' Write Rule

T_{27}	T_{28}
read(Q)	
	write(Q)
write(Q)	

Thomas' Write Rule

T_{27}	T_{28}
read(Q)	
	write(Q)
write(Q)	

Generates **View Serializability**, not conflict serializability.

View Equivalence

Schedules S and S are said to be *view equivalent* if

- For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S , also read the initial value of Q.

View Equivalence

Schedules S and S are said to be *view equivalent* if

- For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S , also read the initial value of Q.

- For each data item Q, if transaction T_i executes $read(Q)$ in schedule S, and if that value was produced by a $write(Q)$ operation executed by transaction T_j , then the $read(Q)$ operation of transaction T_i must, in schedule S , also read the value of Q that was produced by the same $write(Q)$ operation of transaction T_j.

View Equivalence

Schedules S and S are said to be *view equivalent* if

- For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S , also read the initial value of Q.

- For each data item Q, if transaction T_i executes $read(Q)$ in schedule S, and if that value was produced by a $write(Q)$ operation executed by transaction T_j , then the $read(Q)$ operation of transaction T_i must, in schedule S , also read the value of Q that was produced by the same $write(Q)$ operation of transaction T_j.

- For each data item Q, the transaction (if any) that performs the final $write(Q)$ operation in schedule S must perform the final $write(Q)$ operation in schedule S.

View Serializability

A schedule S is *view serializable* if it is view equivalent to a serial schedule.

View Serializability

A schedule S is *view serializable* if it is view equivalent to a serial schedule.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

View Serializability

A schedule S is *view serializable* if it is view equivalent to a serial schedule.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

Blind writes appear in any view-serializable schedule that is not conflict serializable.

Snapshot Isolation

Guarantees that all reads made in a transaction will see a consistent snapshot of the database

Snapshot Isolation

Guarantees that all reads made in a transaction will see a consistent snapshot of the database

The transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.

Snapshot Isolation

Guarantees that all reads made in a transaction will see a consistent snapshot of the database

The transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.

Used by SQL Anywhere, InterBase, Firebird, Oracle, PostgreSQL, MongoDB and Microsoft SQL Server (2005 and later).