Database Management System (DBMS)

Lecture-43

Dharmendra Kumar

January 1, 2021

Graph-Based Protocols

For this type of protocol, we need some prior knowledge of database. To acquire such prior knowledge, we impose a partial ordering $\rightarrow$ on the set D $= \{d_1, d_2, ..., d_n\}$ of all data items. If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access d_i before accessing d_j .

The partial ordering implies that the set D may now be viewed as a directed acyclic graph, called a database graph. Here, we will consider graph with rooted tree. Therefore, we will study tree protocol.

Concurrency Control

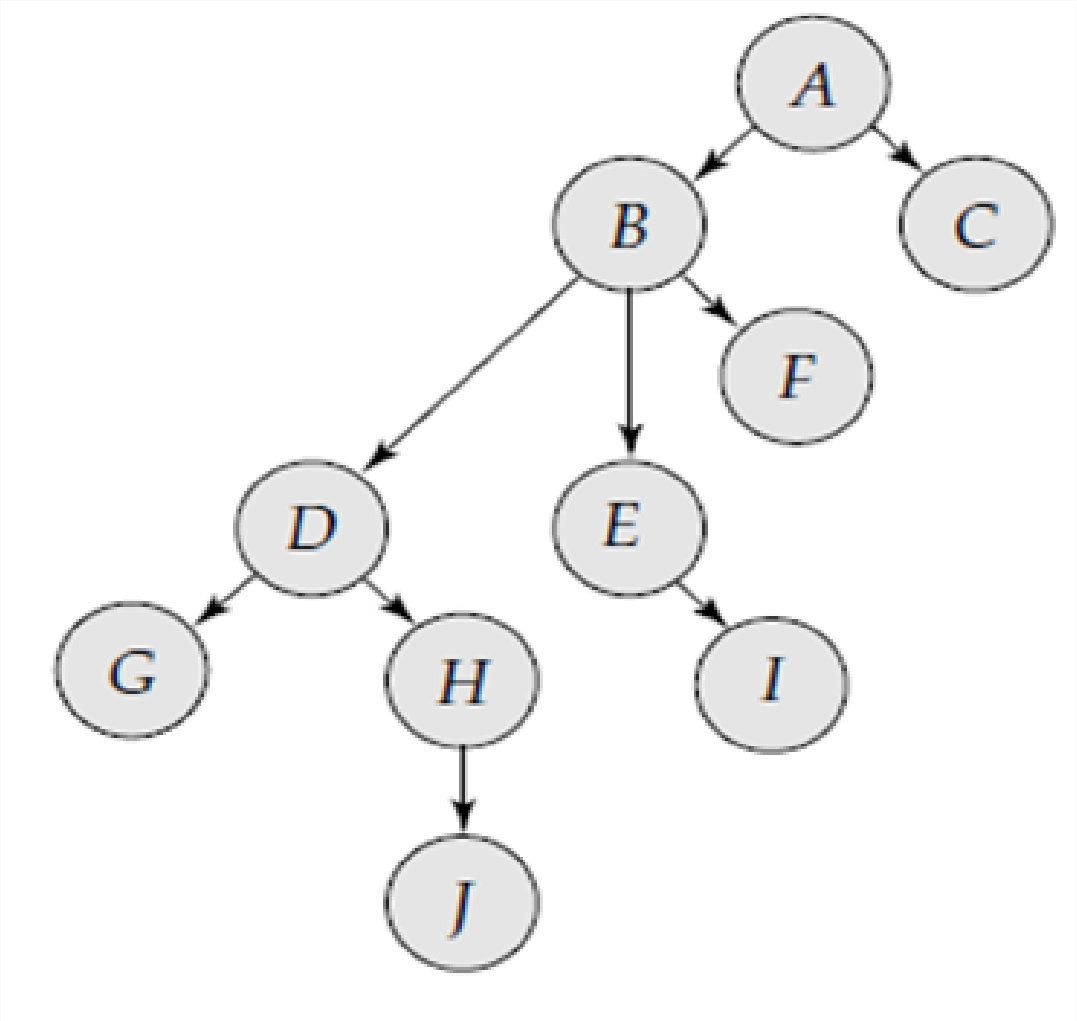
In the **tree protocol,** the only lock instruction allowed is lock-X. Each transaction T_i can lock a data item at most once, and must observe the following rules:

1. The first lock by T_i may be on any data item.
2. Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.
3. Data items may be unlocked at any time
4. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.

All schedules that are legal under the tree protocol are conflict serializabl

Concurrency Control

Example: Consider the database graph of the following figure:-



Concurrency Control

The following four transactions follow the tree protocol on this graph. We show only the lock and unlock instructions:

T_{10}: lock-X(B); lock-X(E); lock-X(D); unlock(B); unlock(E); lock-X(G); unlock(D); unlock(G).

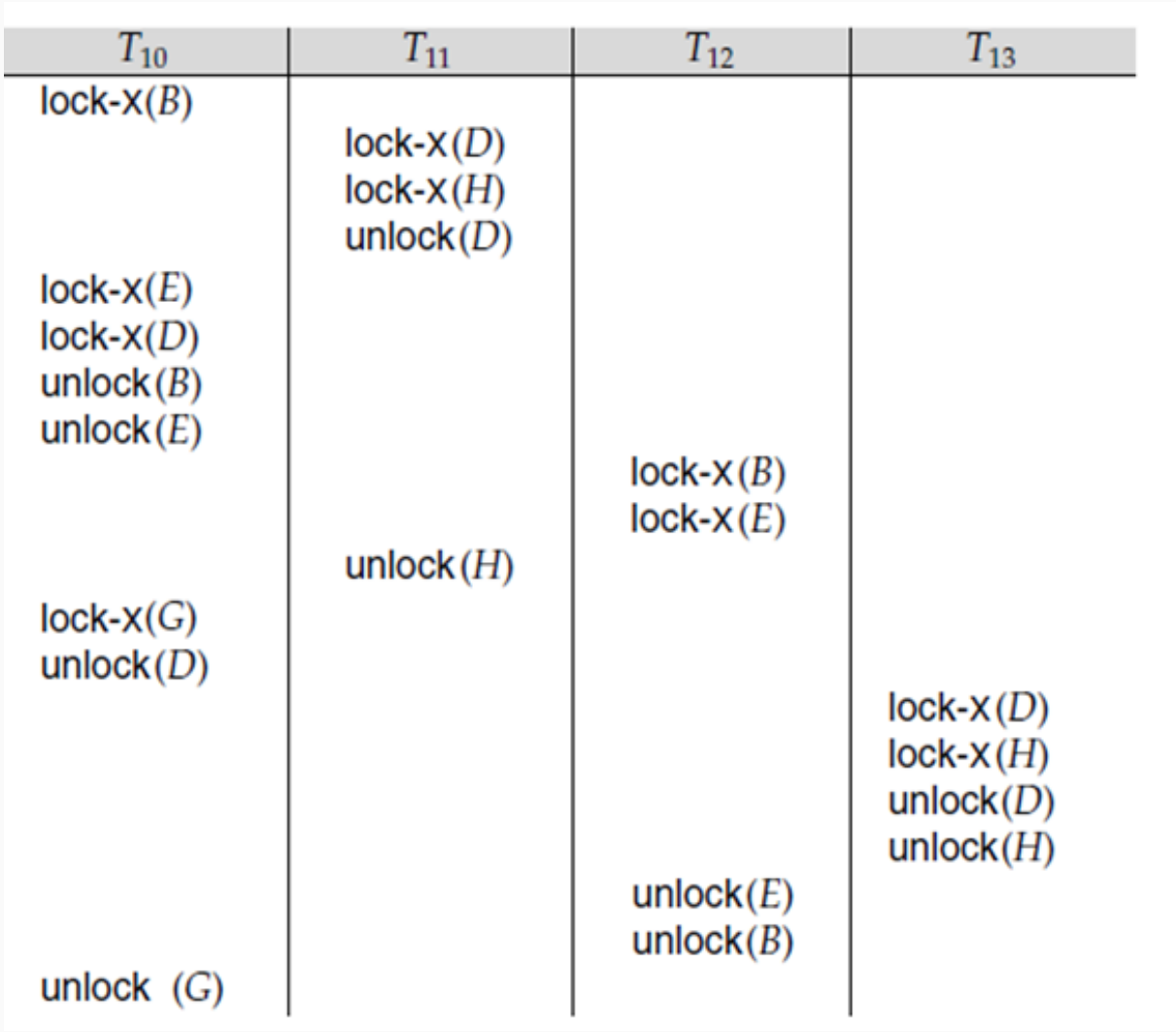
T_{11}: lock-X(D); lock-X(H); unlock(D); unlock(H).

T_{12}: lock-X(B); lock-X(E); unlock(E); unlock(B).

T_{13}: lock-X(D); lock-X(H); unlock(D); unlock(H).

Concurrency Control

One possible schedule in which these four transactions participated appears in the following figure:-



T_{10}	T_{11}	T_{12}	T_{13}
lock-X(B)			
	lock-X(D)		
	lock-X(H)		
	unlock(D)		
lock-X(E)			
lock-X(D)			
unlock(B)			
unlock(E)			
		lock-X(B)	
		lock-X(E)	
	unlock(H)		
lock-X(G)			
unlock(D)			
			lock-X(D)
			lock-X(H)
			unlock(D)
			unlock(H)
		unlock(E)	
		unlock(B)	
unlock (G)			

Observe that the schedule in this figure is conflict serializable. It can be shown not only that the tree protocol ensures conflict serializability, but also that this protocol ensures freedom from deadlock. The tree protocol in this figure does not ensure recoverability and cascadelessness.

Advantage:

1. The tree-locking protocol has an advantage over the two-phase locking protocol in that, unlike two-phase locking, it is deadlock-free, so no rollbacks are required.
2. The tree-locking protocol has another advantage over the two-phase locking protocol in that unlocking may occur earlier. Earlier unlocking may lead to shorter waiting times, and to an increase in concurrency.

Timestamp-Based Protocols

Timestamps

With each transaction T_i in the system, we associate a unique fixed timestamp, denoted by TS(T_i). This timestamp is assigned by the database system before the transaction Ti starts execution. If a transaction T_i has been assigned timestamp TS(T_i), and a new transaction T_j enters the system, then TS(T_i) ¡ TS(T_j).

Concurrency Control

There are two simple methods for implementing this scheme:

1. Use the value of the system clock as the timestamp; that is, a transaction's timestamp is equal to the value of the clock when the transaction enters the system.
2. Use a logical counter that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter when the transaction enters the system.

The timestamps of the transactions determine the serializability order. Thus, if TS(T_i) ¡ TS(T_i), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction T_i appears before transaction T_j

To implement this scheme, we associate with each data item Q two timestamp values:

- **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.

These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

Timestamp-Ordering Protocol

The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

Rule-1: Suppose that transaction T_i issues read(Q).

1. If TS(T_i) $<$ W-timestamp(Q), then the read operation is rejected, and T_i is rolled back.
2. If TS(T_i) $\geq$ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T_i).

Rule-2: Suppose that transaction T_i issues write(Q).

1. If TS(T_i) $<$ R-timestamp(Q), then the system rejects the write operation and rolls T_i back.
2. If TS(Ti) $<$ W-timestamp(Q), then the system rejects this write operation and rolls T_i back.
3. Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(T_i).

If a transaction T_i is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

Concurrency Control

Example: Consider transactions T_{14} and T_{15}. Transaction T_{14} displays the contents of accounts A and B:

T_{14}: read(B);
read(A);
display(A + B).

Transaction T_{15} transfers \$50 from account A to account B, and then displays the contents of both:

T_{15}: read(B);
B := B - 50;
write(B);
read(A);
A := A + 50;
write(A);
display(A + B).

Concurrency Control

Following schedule is possible under timestamp ordering protocol.

T_{14}	T_{15}
read(B)	
	read(B)
	$B := B - 50$
	write(B)
read(A)	
	read(A)
display($A + B$)	
	$A := A + 50$
	write(A)
	display($A + B$)

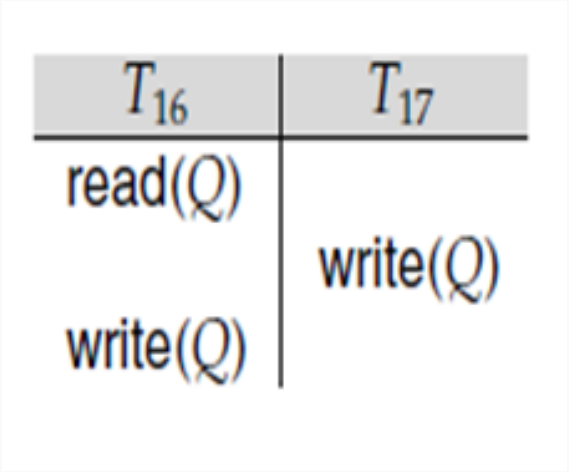
1. The timestamp-ordering protocol ensures conflict serializability.
2. This protocol also ensures freedom from deadlock.
3. There is a possibility of starvation.
4. This protocol can generate schedules that are not recoverable.

Thomas' Write Rule

The modification to the timestamp-ordering protocol, called Thomas' write rule, is this: Suppose that transaction Ti issues write(Q).

1. If TS(T_i) ¡ W-timestamp(Q), then the read operation is rejected, and T_i is rolled back.
2. If TS(T_i) ¡ W-timestamp(Q), then T_i is attempting to write an obsolete value of Q. Hence, this write operation can be ignored.
3. Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(T_i).

Example: Consider following schedule:-



Clearly, this schedule is not conflict serializable and, thus, is not possible under any of two-phase locking, the tree protocol, or the timestamp-ordering protocol. Under Thomas' write rule, the write(Q) operation of T_{16} would be ignored. The result is a schedule that is view equivalent to the serial schedule ¡ T_{16}, T_{17}¿.