Database Systems

Transactions, Concurrency Control and Recovery (Continued - 1)

Jing Sun and Miao Qiao

The University of Auckland

Binary Locks

Two states (values) lock(X) =

- Locked (1) — Item cannot be accessed
- Unlocked (0) — Item can be accessed when requested

Two operations

- lock_item(X)
- unlock_item(X)

- The lock/unlock operations are atom operations.
- Transactions hold the lock on one data item in a mutually exclusive way.
- At most one transaction can hold the lock on an item at a given time — too restrictive for database items

Shared/Exclusive (Read/Write) Locks

Shared/exclusive or read/write locks:

- Read operations on the same item are not conflicting
- Must have an exclusive lock to write

Three locking operations in a transaction T:

- read_lock(X): attempt to request a read lock (denoted as ls(X))
- write_lock(X): attempt to request a write lock (denoted as lx(X))
- unlock(X): release the lock T currently holds on X (denoted as u(X))

Shared/Exclusive (Read/Write) Locks

Shared/exclusive or read/write locks:

- Read operations on the same item are not conflicting
- Must have an exclusive lock to write

		Lock requested	
		S	X
Lock held in mode	S	Yes	No
	X	No	No

Table: The compatibility matrix for shared and exclusive locks.

Shared/Exclusive (Read/Write) Locks

T_1	T_2
$sl_1(A)$; $r_1(A)$;	
	$sl_2(A)$; $r_2(A)$;
	$sl_2(B)$; $r_2(B)$;
$xl_1(B)$ Denied	
	$u_2(A)$; $u_2(B)$
$xl_1(B)$; $r_1(B)$; $w_1(B)$	
$u_1(A)$; $u_1(B)$;	

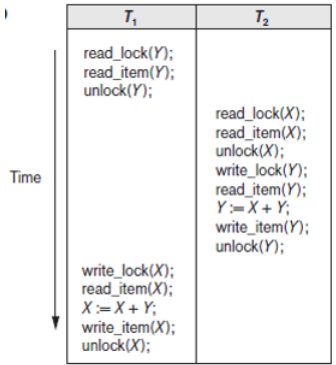
Shared/Exclusive (Read/Write) Locks

- Lock conversion: Transaction that already holds a lock allowed to convert the lock from one state to another
- Upgrading: Issue a read_lock operation then a write_lock operation
- Downgrading: Issue a read_lock operation after a write_lock operation

T_1	T_2
$sl_1(A); r_1(A);$	
	$sl_2(A); r_2(A);$
	$sl_2(B); r_2(B);$
$sl_1(B); r_1(B);$	
$xl_1(B)$ Denied	
	$u_2(A); u_2(B)$
$xl_1(B); w_1(B)$	
$u_1(A); u_1(B);$	

Serializability

Using locking techniques does not guarantee serilizability.



Two-Phase Locking

Two-phase locking (2PL) protocol.

- Expanding (growing) phase — New locks can be acquired but none can be released. Lock conversion can only do upgrades
- Shrinking phase — Existing locks can be released but none can be acquired. Lock conversion can only do downgrades

T_1'
read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);

T_2'
read_lock(X);
read_item(X);
write_lock(Y);
unlock(X)
read_item(Y);
$Y := X + Y$;
write_item(Y);
unlock(Y);

Two-phase locking leads to serializablility

Proof (use mathematical induction). Assume that any schedule of n transactions $T_1, T_2, \cdots, T_n$ under 2PL are serializable (True statement when $n = 1$). Consider a schedule S under 2PL on $n+1$ transactions $T_1, \cdots, T_{n+1}$ — let T_{n+1} be the first transaction with an unlock operation in S (otherwise we can swap the names). Now we prove that S is conflict-equivalent to the schedule $S' = T_{n+1} \cdot [S \setminus T_{n+1}]$ (moving all operations of T_{n+1} to the front of S).
Consider an operation $w_{n+1}(X)$ in T_{n+1}. If there is a conflict operation $o_i(X)$ ($r_i(X)$ or $w_i(X)$) comes before $w_{n+1}(X)$ in schedule S where $i \leq n$ consider the order of $u_i(X)$ and $lx_{n+1}(X)$ in S

- Case 1: $o_i(X) \cdots u_i(X) \cdots lx_{n+1}(X) \cdots w_{n+1}(X) \cdots u_{n+1}(X)$ violates 2PL
- Case 2: $o_i(X) \cdots lx_{n+1}(X) \cdots u_i(X) \cdots$ not compatible

Therefore, no conflict operation of $w_{n+1}(X)$ can appear before $w_{n+1}(X)$ in S. *Please similarly prove that a conflicting operation of $r_{n+1}(X)$ will not appear before $r_{n+1}(X)$ in $S \cdots$.* Therefore, any conflict operation of an operation o in T_{n+1} appears after o in S, it is conflict equivalent to move all operations of T_{n+1} to the front of S.

Two-phase locking leads to serializablility

- If every transaction in a schedule follows the two-phase locking protocol, schedule guaranteed to be serializable
- Two-phase locking may limit the amount of concurrency that can occur in a schedule
- Some serializable schedules will be prohibited by two-phase locking protocol

Variations of Two-Phase Locking

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

- Basic 2PL — Technique described on previous slides
- Conservative (static) 2PL — Requires a transaction to lock all the items it accesses before the transaction begins (Predeclare read-set and write-set)
- Strict 2PL — Transaction does not release exclusive locks until after it commits or aborts
- Rigorous 2PL — Transaction does not release any locks until after it commits or aborts

Locks: Overview

- Binary Locks
- Shared/Exclusive Locks
- Update Locks
- Increment Locks
- Locks with Multiple Granularity

Locking Scheduler

Update Lock

T_1	T_2
$sl_1(A); r_1(A);$	
	$sl_2(A); r_2(A);$
$xl_1(A)$ Denied	
	$xl_2(A)$ Denied

Three lock modes, shared, exclusive and **update**. $ul_i(X)$ grants T_i the right to read X not to write X but only $ul_i(X)$ can be upgraded to $w_i(X)$ while $r_i(X)$ cannot. Update lock looks like a shared lock when we are requesting it and looks like exclusive lock when we already have it. Denote by U the update lock mode.

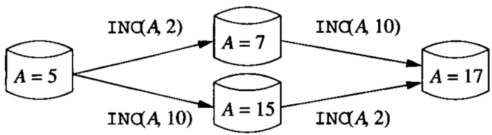
	S	X	U
S	Yes	No	Yes
X	No	No	No
U	No	No	No

T_1	T_2
$ul_1(A); r_1(A);$	
	$ul_2(A); r_2(A);$Denied
$xl_1(A)$	

Increment Locks

Many transactions operate on the database only by incrementing or decrementing stored values. For example, consider a transaction that transfers money from one bank account to another. The useful property of increment actions is that they commute with each other.

- Increment Lock $inc_i(X)$, denoted by I the increment lock mode



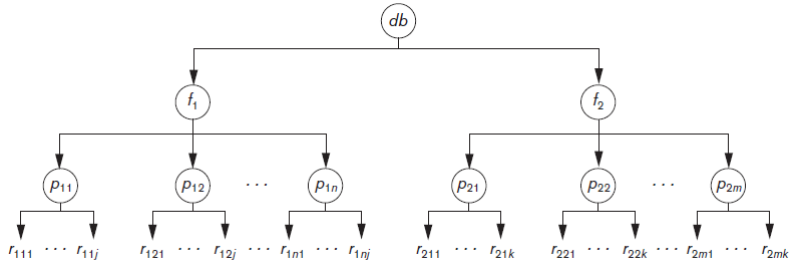
	S	X	I
S	Yes	No	No
X	No	No	No
I	No	No	Yes

Locks: Overview

- Binary Locks
- Shared/Exclusive Locks
- Update Locks
- Increment Locks
- Locks with Multiple Granularity

Locking Scheduler

Locks of Multiple Granularities



Locks of Multiple Granularities

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

Intention locks

Transaction indicates along the path from the root to the desired node, what type of lock (shared or exclusive) it will require from one of the node's descendants

Intention lock types

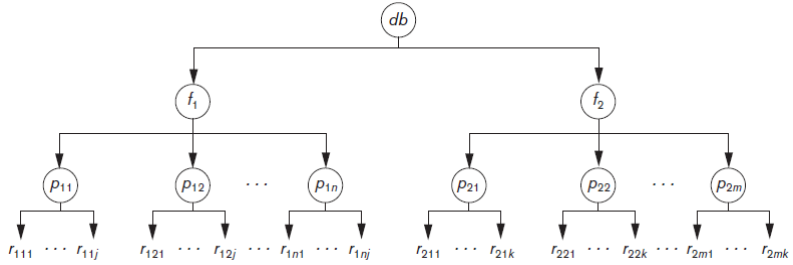
- Intention-shared (IS): Shared locks will be requested on a descendant node
- Intention-exclusive (IX): Exclusive locks will be requested on a descendant node
- Shared-intension-exclusive (SIX): Current node is locked in shared mode but one or more exclusive locks will be requested on a descendant node

Original lock types

- Shared (S): Shared locks is requested on the entire subtree of the node
- Exclusive (X): Exclusive locks is requested on the entire subtree of the node

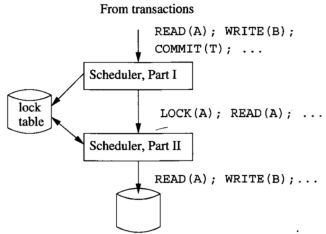
Locks of Multiple Granularities

	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No



Locking Scheduler

- The transactions themselves do not request locks or cannot be relied upon to do so. It is the job of the scheduler to insert lock actions into the stream of reads, writes and other actions that access data.
- Transactions do not release locks. Rather the scheduler releases the locks when the transaction manager tells it that the transaction will commit or abort.



Locking Scheduler

- Part I inserts appropriate lock actions for database access actions (choose lock mode)
- Part II executes the *sequence* of lock and database access actions passed by Part I; assume that the current operation o is on T:
  - If o acquire a lock for a transaction T — look at the lock table
    - if one cannot grant, put the request of T to the lock table, notify Part I to delay T
    - grant the lock to T and update the lock table
  - If keys requested so far by T have been granted and o is a database access operations — execute
  - Commits/aborts a transaction T
    - Notify Part I. Release all locks hold by T. If any transactions are waiting for any of these locks, Part I notifies Part II
  - If notified by Part I that the lock on element X has been released, pick the next transaction T in the waiting list to lock X, notify Part I to resume T.

Thank you for your attention!

Any questions?