LECTURE 08

CONCURRENCY CONTROL
(PART 3/3)

18.4 Locking Systems With Several Lock Modes

Introduction

- The locking scheme of Section 18.3
  - is too simple to be a practical scheme
- <u>Main Problem</u>
  - A transaction *T* must take a lock on a database element *X* even if it only wants to read *X* and not write it
  - There is no reason
    - Why several transactions could not read *X* at the same time, as long as none is allowed to write *X*

Shared and Exclusive Locks

- Motivation
  - The lock we need for writing is “stronger” than the lock we need to read
- Two kinds of locks
  - Shared lock: read lock
  - Exclusive lock: write lock
- For any database element *X*
  - There can be either one exclusive lock on *X*, or no exclusive locks but any number of shared locks

Shared and Exclusive Locks

- Notations

  $sl_i(X)$

  - Transaction T_i requests a shared lock on X

  $xl_i(X)$

  - Transaction T_i requests an exclusive lock on X

  $u_i(X)$

  - Transaction T_i unlocks X; i.e., it relinquishes whatever lock(s) it has on X

Shared and Exclusive Locks

1. *Consistency of transactions*

   (a) A read action $r_i(X)$ must be preceded by $sl_i(X)$ or $xl_i(X)$, with no intervening $u_i(X)$

   (b) A write action $w_i(X)$ must be preceded by $xl_i(X)$, with no intervening $u_i(X)$

   All locks must be followed by an unlock of the same element

Shared and Exclusive Locks

2. *Two-phase locking of transactions*

In any two-phase locked transaction T_i, no action $sl_i(X)$ or $xl_i(X)$ can be preceded by an action $u_i(Y)$, for any Y

Shared and Exclusive Locks

3. *Legality of schedules*

(a) If $xl_i(X)$ appears in a schedule, then there cannot be a following $xl_j(X)$ or $sl_j(X)$, for $j \neq i$, without an intervening $u_i(X)$

(b) If $sl_i(X)$ appears in a schedule, then there cannot be a following $xl_j(X)$, for $j \neq i$, without an intervening $u_i(X)$

Example

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

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**	
	$u_2(A); u_2(B)$
$xl_1(B); r_1(B); w_1(B);$	
$u_1(A); u_1(B);$	

Example

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**	
	$u_2(A);\ u_2(B)$
$xl_1(B);\ r_1(B);\ w_1(B);$	
$u_1(A);\ u_1(B);$	

Legal schedules of consistent, 2PL transactions are conflict-serializable applies to systems with shared and exclusive locks as well

Compatibility Matrices

- Compatibility matrix

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

- The rule for using a compatibility matrix for lock-granting decisions
- We can grant the lock on *X* in mode *C* if and only if for every row *R* such that there is already a lock on *X* in mode *R* by some other transaction, there is a "Yes" in column *C*

Upgrading Locks

- Upgrading locks
  - *T* is first to take a shared lock on *X*
  - Only later when *T* is ready to write the new value, upgrade the lock to exclusive

Example

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

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**	
	$u_2(A); u_2(B)$
$xl_1(B); w_1(B);$	
$u_1(A); u_2(B);$	

Example

- T_1 reads A and B and performs some (possibly lengthy) calculation with them, eventually using the result to write a new value of B

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**	
	$u_2(A);\ u_2(B)$
$xl_1(B);\ w_1(B);$	
$u_1(A);\ u_2(B);$	

Example

- Notice that had T_1 asked for an exclusive lock on B initially, before reading B, then the request would have been denied
- As a result, T_1 finishes later using only an exclusive lock on B than it would if it used the upgrading strategy

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**	
	$u_2(A);\ u_2(B)$
$xl_1(B);\ w_1(B);$	
$u_1(A);\ u_2(B);$	

Example

- Unfortunately, indiscriminate use of upgrading introduces a new and potentially serious source of deadlock.

- Suppose, that T_1 and T_2 each read and write A
- If both transactions use an upgrading approach

T_1	T_2
$sl_1(A)$	
	$sl_2(A)$
$xl_1(A)$ **Denied**	
	$xl_2(A)$ **Denied**

Update Locks

- We can avoid the deadlock problem in the previous example with update locks
- An update lock $ul_i(X)$
  - gives transaction T_i only the privilege to read X, not to write X
  - However, only the update lock can be upgraded to a write lock later; a read lock cannot be upgraded

Update Locks

- An asymmetric compatibility matrix
  – The update lock looks like a shared lock when we are requesting it, and
  – looks like an exclusive lock when we already have it

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

Example

- What will happen if $sl_1(B)$ => $ul_1(B)$?

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**	
	$u_2(A);\ u_2(B)$
$xl_1(B);\ w_1(B);$	
$u_1(A);\ u_2(B);$	

Example

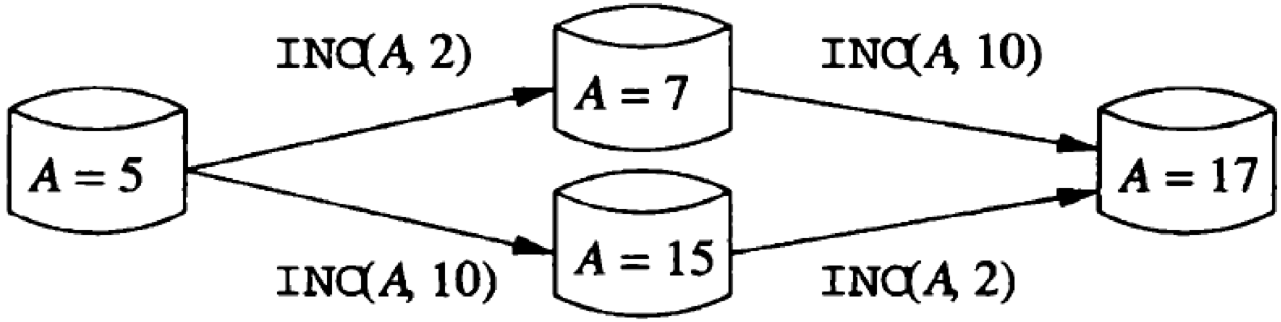
T_1: $ul_1(A); r_1(A); xl_1(A); w_1(A); u_1(A);$
T_2: $ul_2(A); r_2(A); xl_2(A); w_2(A); u_2(A);$

T_1	T_2
$ul_1(A); r_1(A);$	
	$ul_2(A)$ **Denied**
$xl_1(A); w_1(A); u_1(A);$	
	$ul_2(A); r_2(A);$
	$xl_2(A); w_2(A); u_2(A);$

The lock system has effectively prevented concurrent execution of T_1 and T_2

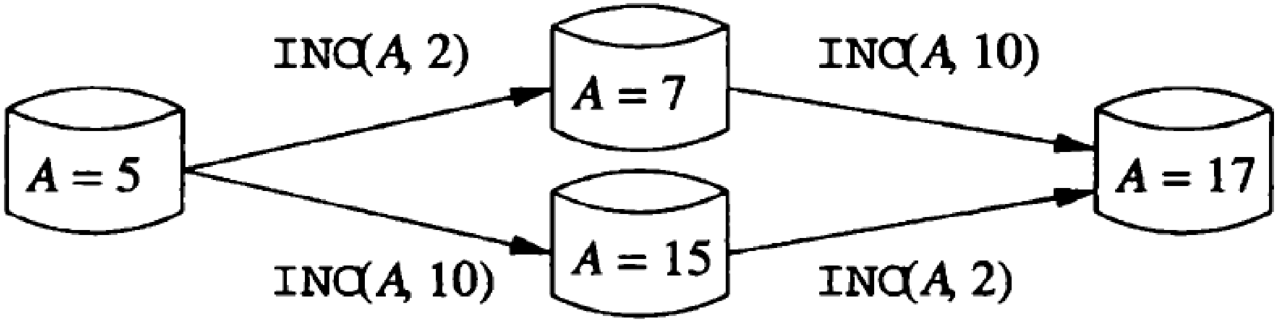
Increment Locks

- Motivation
  - 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
- Increment actions commute with each other



Increment Locks

- Increment actions commute with each other
- Increment actions commute with neither reading nor writing



Increment Locks

- Increment action: INC(A, c)
  - Stand for the atomic execution of the following steps
    READ(A, t); t := t+c; WRITE(A, t);
- Increment lock

  $il_i(X)$
  T_i requesting an increment lock on X

  $inc_i(X)$
  T_i increments X by some constant

Increment Locks

- Changes
  1. A consistent transaction can only have an increment action on X if it holds an increment lock on X at the time. An increment lock does not enable read or write actions, however
  2. In a legal schedule, any number of transactions can hold an increment lock on X at any time. However, if an increment lock on X is held by some transaction, then no other transaction can hold either a shared or exclusive lock on X at the same time

Increment Locks

- Changes
  - The action $inc_i(X)$ conflicts with both $r_j(X)$ and $w_j(X)$, for $j \neq i$, but does not conflict with $inc_j(X)$
- Compatibility matrix

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

Example

T_1: $sl_1(A);\ r_1(A);\ il_1(B);\ inc_1(B);\ u_1(A);\ u_1(B);$
T_2: $sl_2(A);\ r_2(A);\ il_2(B);\ inc_2(B);\ u_2(A);\ u_2(B);$

T_1	T_2
$sl_1(A);\ r_1(A);$	
	$sl_2(A);\ r_2(A);$
	$il_2(B);\ inc_2(B);$
$il_1(B);\ inc_1(B);$	
	$u_2(A);\ u_2(B);$
$u_1(A);\ u_1(B);$	

…The End of This Lecture…

Q&A