Washington University in St. Louis

Engineering

Database Management Systems

- Concurrency Control

Doug Shook

Review

- Why do we need transactions?
  - What does a transaction contain?
  - What are the four properties of a transaction?

- What is a schedule?
  - What is a conflict?
  - What does it mean for a schedule to be serializable?

Locking

- We wish to create serializable schedules

- Many techniques for this:
  - Locking
  - Timestamps
  - Multiversion

- In practice, locking is used most frequently
  - Operates on data items
  - What is a data item?
  - Many types of locks available

Binary Locks

- Two states:
  - Locked/unlocked
  - Cannot access an item while it is locked
- Enforces mutual exclusion

Binary Locks

```
lock_item(X):
B:  if LOCK(X) = 0                   (* item is unlocked *)
        then LOCK(X) ←1              (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                     (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Implementation

- Must maintain a table of locks
  - And a queue for transactions that are waiting
- Must obey the following rules:
  - Must lock before any read or write
  - Must unlock after any read or write
  - Transaction cannot lock or unlock an item if it is already locked/unlocked
- In practice, while binary locks work just fine, they are considered too restrictive
  - Why?

Shared/Exclusive Locks

- We wish for our locks to mesh well with conflicts
  - Is it a problem for two transactions to read the same item?
- Two kinds of locks:
  - Read
  - Write
- Only one write lock at a time
  - Multiple read locks can exist for the same item
  - How to track them?

Shared/Exclusive Properties

- The following rules must be obeyed:
  - Must acquire a read or write lock before reading
  - Must acquire a write lock before writing
  - A transaction cannot reacquire a lock that it already has
- It is possible to convert locks
  - Read → Write
  - Write → Read
- What must be true for conversion to be allowed?

Two Phase Locking

- All locks should be acquired before the first unlock statement
  - Growing phase: increase number of locks
  - Shrinking phase: decrease number of locks

- Guarantees serializability

Two phase locking

T_1	T_2

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

$T_1{}'$

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);

Types of Two Phase

- Basic
- Conservative
  - Must acquire all locks before beginning the transaction
- Strict
  - Do not release any write locks until after commit or abort
  - Leads to strict schedules
    - No transaction can read or write X until the last transaction that wrote X has committed

Issues With Two Phase

- Cannot generate all possible serializable schedules
- Reduces concurrency
- Can lead to deadlock
- Can lead to starvation

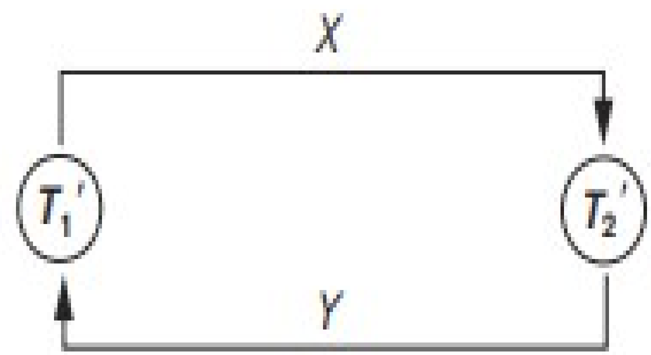
Deadlock

(a)

Time

T_1'	T_2'
read_lock(Y); read_item(Y);	
	read_lock(X); read_item(X);
write_lock(X);	
	write_lock(Y);

(b)



Deadlock Prevention

- Timestamp based
- Wait-die: older transaction is allowed to wait, younger transaction must abort and restart
- Wound-wait: Older transaction can abort younger transaction, restarting it later. Younger transactions simply wait.
- Can lead to unnecessary aborts

Deadlock Prevention

- Non-timestamp based
- No wait – if lock is unavailable, abort
- Cautious wait – If the transaction that has the lock is not waiting on another lock, then wait. Otherwise abort.

Deadlock Detection

- Create a graph
- Nodes are transactions
- Directed edges from transactions that wait on locks from other transactions
  - Drop edges when waiting is over
  - When do we know deadlock has occurred?
  - What do we do?
- Works best with small transactions
  - Why?

Starvation

- Occurs when a transaction has to wait a very long time.
  - Possibly indefinitely?
  - When does this happen?

- Solutions:
  - Time scaling priority
  - FCFS

Granularity

- What should we be locking?
  - Fields?
  - Columns?
  - Pages?
  - Tables?

- How does granularity affect concurrency?

- Why not just lock everything on the field level?

- What is the best size?

Granularity

- Two transactions: one wants to update all records in a file, the other wants to read a single record
  - How does order affect locking?

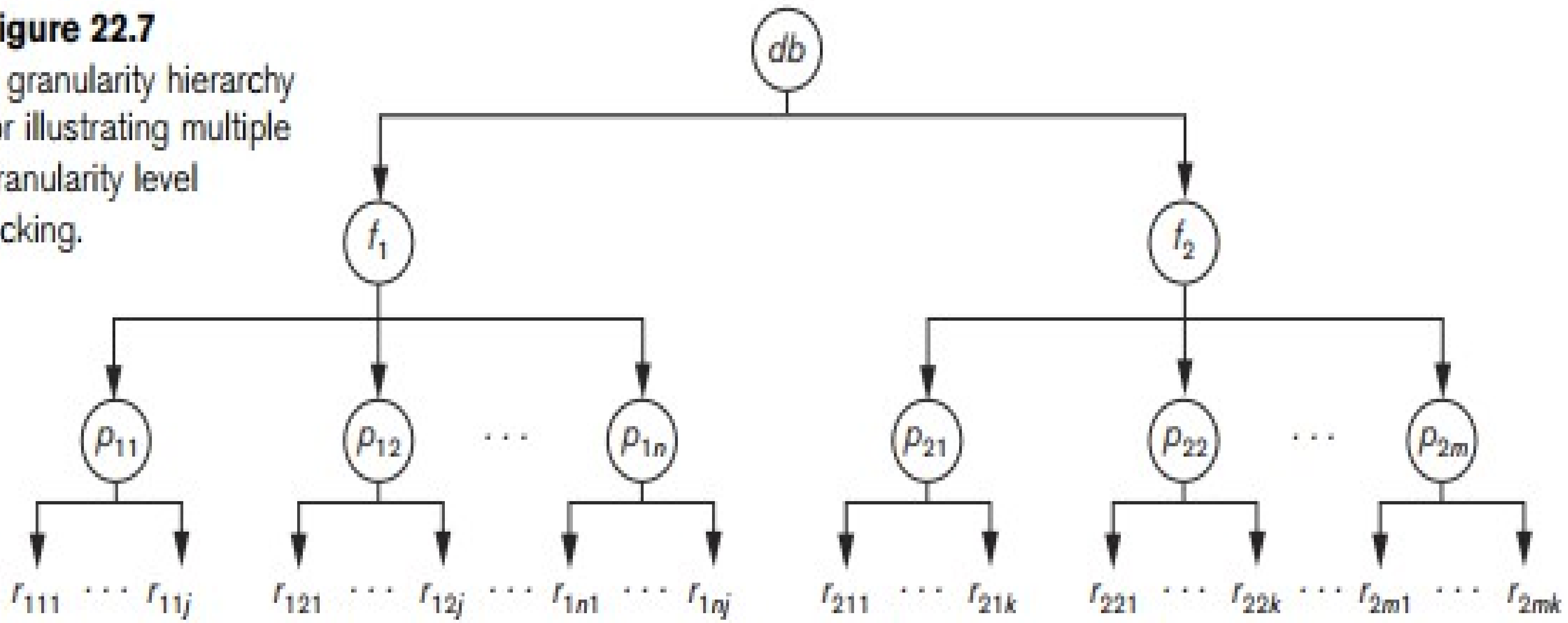


Figure 22.7
A granularity hierarchy for illustrating multiple granularity level locking.

Intention Locks

- What locks would be requested on descendants of the tree?
- Intention Shared
- Intention Exclusive
- Shared intention exclusive
  – Currently share-locked, intends to be exclusive

Intention Locks

- Lock the root first
- Can be locked in S or IS mode if parent is locked in IS or IX mode.
- Can be locked in X, IX, or SIX mode if parent is locked in IX or SIX mode
- Nodes can only be unlocked if children are unlocked

Intention Locks

	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

Intention Locks

- So for our previous example:
  - T1 locks root and file with IX
  - T2 may still lock records with IS/S while file is in IX

- If T2 goes first: root, file, record locked with IS/S
  - T1 cannot acquire lock until T2 is done
  - But the check is a lot faster!