Database Management System – 42 (Concurrency Control Techniques)

Ajay James
Asst. Prof in CSE
Government Engineering College Thrissur

Outline

- Binary Locks
- Shared/exclusive locks
- Conversion of locks
- Two Phase locking
- Variations of 2-phase locking

Introduction

- Lock
  - Variable associated with a data item describing status for operations that can be applied
  - One lock for each item in the database
  - used as a means of synchronizing the access by concurrent transactions to the database items

Binary locks

- Two states (values)
  - Locked (1)
    - Item cannot be accessed
  - Unlocked (0)
    - Item can be accessed when requested

```
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;
```

Binary locks

- Lock table specifies items that have locks
- Lock manager subsystem
  - Keeps track of and controls access to locks
  - Rules enforced by lock manager module
- At most one transaction can hold the lock on an item at a given time
- Binary locking too restrictive for database items

Shared/exclusive or read/write locks

- Read operations on the same item are not conflicting
- Must have exclusive lock to write
- Multiple-mode lock
- Three locking operations
  - read_lock(X)
  - write_lock(X)
  - unlock(X)

Shared/exclusive or read/write locks

- Read-locked item is also called share-locked
  - Because other transactions are allowed to read the item
- Write-locked item is called exclusive-locked
  - Because a single transaction exclusively holds the lock on the item

Shared/exclusive or read/write locks

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
            no_of_reads(X) ← 1
            end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

Shared/exclusive or read/write locks

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) −1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```


Rules for locking

1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T
2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.
3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.

Rules for locking

4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
5. A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X.
6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.

Conversion (Upgrading, Downgrading) of 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

Two-Phase Locking

- Two-phase locking protocol
  - All locking operations (read_lock, write_lock) precede the first unlock operation in the transaction
- Divided into two phases
- ***Expanding or growing(first) phase***
  - during which new locks on items can be acquired but none can be released
- ***Shrinking (second) phase***
  - during which existing locks can be released but no new locks can be acquired
- If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase
- Downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase

No 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);

Two-phase locking

T_1'	T_2'

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

Guaranteeing Serializability by Two-Phase Locking

- 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

- 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
  - Deadlock-free protocol

Variations of Two-Phase Locking

- 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

Reference

- Elmasri R. and S. Navathe, Database Systems: Models, Languages, Design and Application Programming, Pearson Education 6th edition and 7th edition

Thank you