Database System

Concurrency Control

Muhammad Tariq Mahmood
tariq@koreatech.ac.kr
School of Computer Science and Engineering
Korea University of Technology and Education

Database Concurrency Control

- Purpose of Concurrency Control
  - To enforce Isolation (through mutual exclusion) among conflicting transactions.
  - To preserve database consistency through consistency preserving execution of transactions.
  - To resolve read-write and write-write conflicts.

- Example:
  - In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Concurrency Control Algorithms

- Lock based CC Algorithms
  - Two-phase locking Algorithm
  - Multiversions CC algorithms
  - Validation CC Algorithms
  - ….

- Timestamp based CC Algorithms
  - Basic Timestamp based Algorithms
  - …….

Two-Phase Locking Techniques

- Locking is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction.
- Example:
  - Lock (X). Data item X is locked in behalf of the requesting transaction.
- Unlocking is an operation which removes these permissions from the data item.
- Example:
  - Unlock (X): Data item X is made available to all other transactions.
- Lock and Unlock are Atomic operations.

Two-Phase Locking Techniques: Essential components

- Two locks modes:
  - (a) shared (read)
  - (b) exclusive (write).
- Shared mode: shared lock (X)
  - More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X)
  - Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.
- Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Two-Phase Locking Techniques: Essential components

- Lock Manager:
  - Managing locks on data items.

- Lock table:
  - Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Two-Phase Locking Techniques: Essential components

- Database requires that all transactions should be well-formed. A transaction is well-formed if:
  - It must lock the data item before it reads or writes to it.
  - It must not lock an already locked data items and it must not try to unlock a free data item.

```
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;
```

Figure 22.1
Lock and unlock operations for binary 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;
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;
```

Figure 22.2
Locking and unlocking operations for two-mode (read-write or shared-exclusive) locks.

Two-Phase Locking Techniques: The algorithm

- If the simple binary locking scheme described here is used, every transaction must obey the following rules:

1. A transaction *T* must issue the operation lock_item(*X*) before any read_item(*X*) or write_item(*X*) operations are performed in *T*.
2. A transaction *T* must issue the operation unlock_item(*X*) after all read_item(*X*) and write_item(*X*) operations are completed in *T*.
3. A transaction *T* will not issue a lock_item(*X*) operation if it already holds the lock on item *X*.1
4. A transaction *T* will not issue an unlock_item(*X*) operation unless it already holds the lock on item *X*.

Two-Phase Locking Techniques: The algorithm

- When we use the shared/exclusive locking scheme, the system must enforce the following rules:

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.*3
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*. This rule may be relaxed, as we discuss shortly.
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*. This rule may also be relaxed, as we discuss shortly.
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*.

Two-Phase Locking Techniques: The algorithm

- Two Phases:
  - (a) Locking (Growing)
  - (b) Unlocking (Shrinking).

- **Locking (Growing) Phase:**
  - A transaction applies locks (read or write) on desired data items one at a time.

- **Unlocking (Shrinking) Phase:**
  - A transaction unlocks its locked data items one at a time.

- **Requirement:**
  - For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Two-Phase Locking Techniques: The algorithm

- Transactions *T1* and *T2* in Figure do not follow the two-phase locking protocol because the write_lock(*X*) operation follows the unlock(*Y*) operation in *T1*, and similarly the write_lock(*Y*) operation follows the unlock(*X*) operation in *T2*.

(a)

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);

(b) Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

Two-Phase Locking Techniques: The algorithm

(c)

Time

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);	

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

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

Result of schedule S:
X=50, Y=50
(nonserializable)

Figure 22.3
Transactions that do not obey two-phase locking. (a) Two transactions T_1 and T_2. (b) Results of possible serial schedules of T_1 and T_2. (c) A nonserializable schedule S that uses locks.

Two-Phase Locking Techniques: The algorithm

- If we enforce two-phase locking, the transactions can be rewritten as *T*’1 and *T*’2, as shown in Figure

- Now, the schedule shown in Figure 22.3(c) is not permitted for *T*’1 and *T*’2 (with their modified order of locking and unlocking operations) under the rules of locking

Figure 22.4
Transactions T_1' and T_2', which are the same as T_1 and T_2 in Figure 22.3, but follow the two-phase locking protocol. Note that they can produce a deadlock.

T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y)	unlock(X)
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Two-Phase Locking Techniques: The algorithm

- Two-phase policy generates two locking algorithms
  - (a) **Basic**
  - (b) **Conservative**
- **Conservative**:
  - Prevents deadlock by locking all desired data items before transaction begins execution.
- **Basic**:
  - Transaction locks data items incrementally. This may cause deadlock which is dealt with.
- **Strict**:
  - A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

Two-Phase Locking Techniques: The algorithm

- Two-phase policy generates two locking algorithms
  - (a) **Basic**
  - (b) **Conservative**

- **Conservative**:
  - Prevents deadlock by locking all desired data items before transaction begins execution.

- **Basic**:
  - Transaction locks data items incrementally. This may cause deadlock which is dealt with.

- **Strict**:
  - A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- **Deadlock** occurs when *each* transaction *T* in a set of *two or more transactions* is waiting for some item that is locked by some other transaction *T* ‘in the set.

- A simple example is shown in Figure where the two transactions *T’*1 and *T’*2 are deadlocked in a partial schedule;

- *T’*1 is in the waiting queue for *X*, which is locked by *T’*2, while *T’*2 is in the waiting queue for *Y*, which is locked by *T’*1.

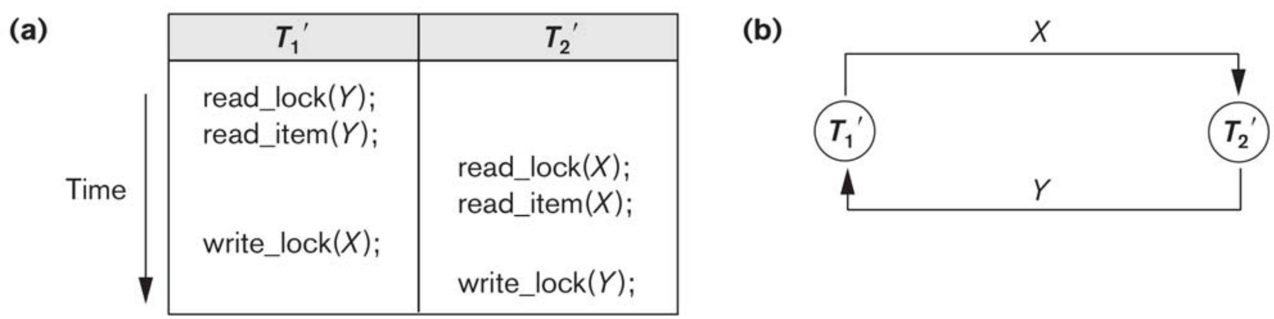

Figure 22.5
Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- Deadlock prevention
  - A transaction locks all data items it refers to before it begins execution.
  - This way of locking prevents deadlock since a transaction never waits for a data item.
  - The conservative two-phase locking uses this approach.

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- Deadlock detection and resolution
  - In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.
  - A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle. One of the transaction o

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- Deadlock avoidance
  - There are many variations of two-phase locking algorithm.
  - Some avoid deadlock by not letting the cycle to complete.
  - That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
  - Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- Starvation
  - Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.
  - In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back.
  - This limitation is inherent in all priority based scheduling mechanisms.
  - In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.

Two-Phase Locking Techniques: Dealing with Deadlock and Starvation

- Timestamp
  - A monotonically increasing variable (integer) indicating the age of an operation or a transaction.
  - A larger timestamp value indicates a more recent event or operation.
  - Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.