UNIT 6

Concurrency Control

Content

- Types of locks and system lock tables
- Serializablility by Two-Phase Locking
- Dealing with Deadlock and Starvation
- Timestamp ordering
- Validation concurrency control

Types of Lock

- Some of the main technique used to control concurrent execution of transaction are based on the concept of locking data items.
- Types of Lock
  - Binary Lock :
  - Shared/Exclusive (Read/Write) Lock

Binary Lock

- A Binary Lock can have two state or values:
  - Lock
  - Unlock
- 1 -> indicates locked
- 0 -> indicates unlocked.

Locking

- **Locking** is an operation which secures (a) permission to Read or (b) permission to Write a data item for a transaction.
- A distinct lock is associated with each database item x.
- A value of lock(x)=1
  - A value x can not be accessed by database operation that request the item.
- Example: Lock (X)
  - Data item X is locked in behalf of the requesting transaction.

Unlocking

- **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.

Exclusive/Shared Lock

- 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/Shared Lock

- **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.
  - If transaction have exclusive lock then no transaction have exclusive lock on same item.

Lock table and Lock manager

- **Lock table :**
  - The system need to maintain only these record for the item that are concurrently locked in a Lock table.
  - Lock table is organize as a hash file.
  - Item not in lock table are consider to be unlocked.

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

- **Lock Manager :**
  - Managing locks on data items.

- 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 Operation of Binary Lock

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)

the lock manager wakes up the transaction;

goto B

end;

unlock Operation of Binary Lock

unlock_item(X) :

LOCK (X) ← 0 (*unlock the item*)

if any transactions are waiting then

wake up one of the waiting the transactions;

Rules for transaction in binary locking scheme

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

Lock Operation of Shared Lock

read_item(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;
```

Lock Operation of exclusive Lock

write_item(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 Operation for two-mode

Unlock(x) :

```
if LOCK (X) = "write-locked" then
        begin
                LOCK (X) ← "unlocked";
                wake up on of the waiting transaction, if any
        end
 else if LOCK (X) = "read-locked" then
         begin
                no_of_read(X) ← no_of_read(X)-1 ;
                if no_of_read(X) =0 then
                        begin
                            LOCK (X) ← "unlocked";
                            wake up on of the waiting transaction, if any
                        end
         end;
```

Rules for transaction in shared/exclusive locking scheme

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

Rules for transaction in shared/exclusive locking scheme

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 a 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.

Serializablility by Two Phase Locking

- A transaction is said to follow the **two-phase locking protocol** if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction.
- Such transaction can be divided in to 2 phase
  - Expanding or Growing (first) Phase
  - Shrinking (Second) Phase

Serializablility by Two Phase Locking

- **Locking (Growing) Phase:** In this phase new locks on the items can be acquired but none can be released.
  - A transaction applies locks (read or write) on desired data items one at a time.(Gain Lock)
- **Unlocking (Shrinking) Phase:** In this phase existing lock can be released but no new lock can be acquired.
  - A transaction unlocks its locked data items one at a time.(Release Lock)
- **Requirement:** For a transaction these two phases must be mutually exclusively, that is,
  - During locking phase unlocking phase must not start
  - During unlocking phase locking phase must not begin.

Serial Schedule of T1 and T2

T1	T2	Result
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2
write_lock (X);	Write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Problem to violated to phase locking Protocol

T1	T2
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 :

X=50; Y=50

Non-Serializable because it violated two phase policy.

T1	T2
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);

- Follows Two phase locking protocols

Two phase locking protocols

- **Advantage:**
  - It produce Serializable schedule.
- **Problem With two phase locking protocols**
  - Two phase locking protocols can produce deadlock.

Lock conversion

- **Lock upgrade:** existing read lock to write lock

  if Ti has a read-lock (X) and Tj has no read-lock (X) (i ≠ j) **then**

  convert read-lock (X) to write-lock (X)

  else

  force Ti to wait until Tj unlocks X

- **Lock downgrade:** existing write lock to read lock

  Ti has a write-lock (X) (*no transaction can have any lock on X*)

  convert write-lock (X) to read-lock (X)

Variation of two phase locking

- Two-phase policy generates two locking algorithms
  - Basic
  - Conservative.
- **Conservative:**
  - Prevents deadlock by locking all desired data items before transaction begins execution by pre-declaring its read-set and write-set.
  - Deadlock free protocol
  - If any of predeclared items needed is can not be locked, the transaction does not lock any of its items, and it wait until all the item available for locking.

Terms in two-phase locking protocols

- **Strict 2PL** :
  - A transaction must released all it's exclusive lock only when it is committed or abort.
  - Transaction T does not released any of its exclusive lock until after commit or aborts.
- **Rigorous 2PL :**
  - A transaction must released all it's lock only when it's commit or abort.
  - Transaction T does not released any of its lock(exclusive or Shared) until after commit or aborts.
  - It is easier to implement than strict 2PL.

Difference between Conservative and Rigorous 2PL

- Conservative: Growing phase
- Rigorous : Expanding Phase

Deadlock

- **Deadlock** occurs when each transaction T in a set of two or more transaction is waiting for some item that is locked by some other transaction T' in the set.
- Hence each transaction in the set is on a waiting queue, waiting for one of the other transaction in the set to release the lock on an item.

Deadlock

T1	T2
read_lock (Y) read_item(Y)	
	read_lock (X) read_item(X)
write_lock (X) (waits for X)	
	write_lock (Y); (waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

3. Dealing with Deadlock

- There are several ways to dealing with deadlock
  - Deadlock prevention
  - Deadlock detection
  - Prevent Starvation

Deadlock prevention

- ***Deadlock prevention*** protocols ensure that the system will *never* enter into a deadlock state.
- Some prevention strategies :
  1. Require that each transaction locks all its data items before it begins execution (pre-declaration).
     - The conservative two-phase locking uses this approach
  2. On the basis of transaction timestamp TS(T)

Transaction Timestamp TP(S)

- Transaction timestamp is a unique identifier assigned to each transaction.
- It is based on the order in which transaction are started.
- If transaction T_1 starts before transaction T_2 then

  $TS(T_1) < TS(T_2)$

- Older transaction has smaller timestamp value.
- Two scheme that prevent deadlock
  1. Wait-Die
  2. Wound-Wait

Deadlock prevention Scheme.

- **wait-die scheme** — non-preemptive

  If TS(i)< TS(j), then (Ti older than Tj)

  Ti allowed to wait;

  otherwise (Ti younger than Tj)

  abort Ti(Ti dies) and

  restart it later with the same timestamp.

  - Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
  - A transaction may die several times before acquiring needed data item

- **wound-wait scheme** — preemptive

  If TS(i)< TS(j), then (Ti older than Tj)

  abort Tj (Ti wounds Tj)

  restart it later with the same timestamp.

  otherwise (Ti younger than Tj)

  Ti allowed to wait

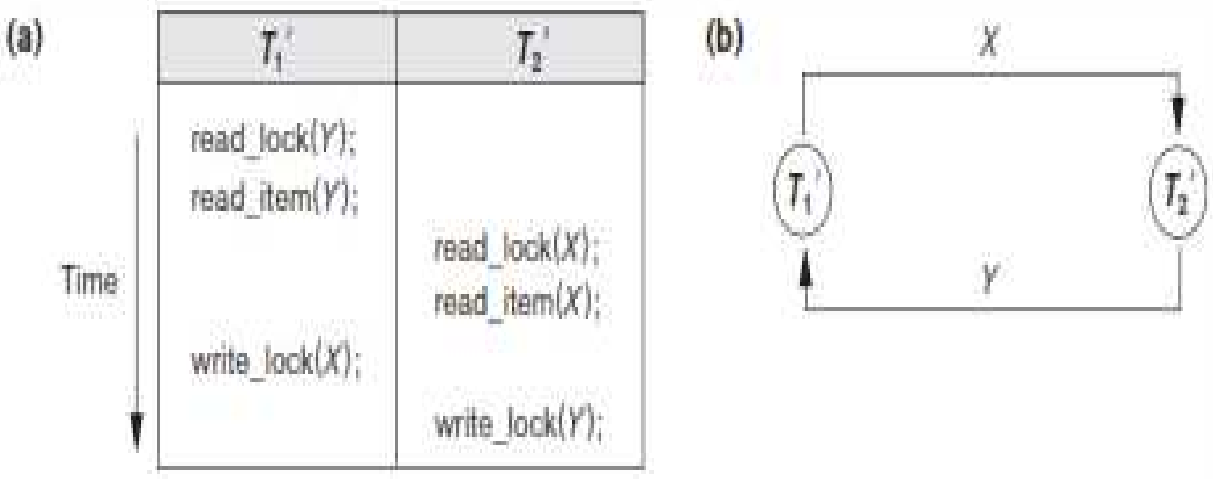
  - older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
  - may be fewer rollbacks than *wait-die* scheme.

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 of the cycle is selected and rolled back.

wait-for graph

- One node is created in the wait-for graph for each transaction that is currently executing.
- Whenever transaction Ti waiting to lock an item X, that is currently lock by transaction Tj.
  - Direct edge is created from (Ti ⟶ Tj)
- We have a state of deadlock if and only if the wait-for graph has a cycle.



Check whether the following schedule produce deadlock or not?

T1	T2

lock (A) ;
read (A) ;
write(A);
 	
lock(B);
	
	lock(B);
	write(B);
	read(B);
	
	lock(A);

Apply two phase locking and then check whether schedule is deadlocked or not?	
T1	T2

read (A) ;
 	
read(B);
If A=1
then B:=B+1
 	
Write(b)
	
	read(B);
	Read(A)
	
	If B=1 then
	A:=A+1
	Write(A)

Starvation

- Starvation :it occurs when a transaction cannot proceed for an indefinite period of time while other transaction in the system continue normally.
- This may occur if the waiting scheme for locked items is unfair.
- Solution of Starvation :
  - First come first served
  - Based on priority
    - But increased the priority of the transaction the longer it waits. until it get highest priority.

Concurrency control based on timestamp ordering

Timestamp

- Transaction timestamp is a unique identifier assigned to each transaction.
- It is based on the order in which transaction are started.
  - A larger timestamp value indicates a more recent event or operation.
  - Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

Timestamp ordering algorithm

- **Timestamp Ordering** : A schedule in which the transaction participate is then Serializable, and equivalent serial schedule has the transaction in order of their timestamp value this is called timestamp ordering.
- order of accessed item that does not violate serializablility order, for that algorithm associated with each database item X is used.

- **read_TS(X)**
  - The read timestamp of item X; this is largest timestamp among all the timestamp of transaction that have successfully read item X.
  - read_TS(X)=TS(T)
    - Where T is the youngest transaction that has read X successfully.
- **write_TS(X)**
  - The write timestamp of item X; this is largest timestamp among all the timestamp of transaction that have successfully written item X.
  - write_TS(X)=TS(T)
    - Where T is the youngest transaction that has written X successfully.

Timestamp based concurrency control algorithm

Basic Timestamp Ordering

1. Transaction T issues a write_item(X) operation:
   a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then

      abort and roll-back T and reject the operation.

      (younger transaction has already read the data item)

   b. If the condition in part (a) does not exist, then

      execute write_item(X) of T and set write_TS(X) to TS(T).

Timestamp based concurrency control algorithm

Basic Timestamp Ordering

2. Transaction T issues a read_item(X) operation:

   a. If write_TS(X) > TS(T), then

      abort and roll-back T and reject the operation.

      (younger transaction has already written to the data item)

   b. If write_TS(X) ≤ TS(T), then

      execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

- **Advantage :**
  - Basic TO always produce conflict Serializable schedule.
  - Deadlock does not occur.
- **Problem with Basic TO**
  - Cyclic restart may occur if a transaction is continually aborted and restarted.
- A Modification of Basic TO algorithm is known as **Thomas's write rule.**
  - It does not enforce conflict serializability
  - But it reject fewer write operation by modifying the checks for the write_item(X) operation.

Timestamp based concurrency control algorithm

Thomas's Write Rule

1. If read_TS(X) > TS(T) then abort and roll-back T and reject the operation.
2. If write_TS(X) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
3. If the conditions given in 1 and 2 above do not occur, then execute write_item(X) of T and set write_TS(X) to TS(T).

Validation concurrency control techniques

- optimistic concurrency control technique (validation or certification technique)
  - No checking is done while the transaction is executing.
  - It follows some scheme.
    - In this scheme , updation in the transaction are not applied directly to the database item until the transaction reaches it end.
    - All updation applied to local copies of data item that are kept for transaction.
    - At the end of transaction, concurrency control phases checks whether any of transaction's updates violates serializability or not.

Three phases of concurrency control protocol

- **Read Phase:**
  - A transaction can read values of committed data items from the database.
  - However , updation are applied only to local copies of data items.
- **Validation Phase :**
  - Checking is performed to ensure that serializability will not be violated if transaction updates are applied to the database.
- **Write Phase**
  - If validation phase is successful, the transaction updates are applied to the database;
  - Otherwise , the updates are discarded and the transaction is restarted .

Validation Phase

- This phase for Ti checks that, for each transaction Tj that is either committed or is in its validation phase, one of the following conditions holds:

  1. Tj completes its write phase before Ti starts its read phase.
  2. Ti starts its write phase after Tj completes its write phase, and the read_set of Ti has no items in common with the write_set of Tj
  3. Both the read_set and write_set of Ti have no items in common with the write_set of Tj, and Tj completes its end phase.

End of the Syllabus
Thank You

ALL THE BEST