Database Management Systems

Transaction Management

Scheduler Protocols

How to Generate Desired Schedules

- Locking based protocols
- Timestamp based protocols
- Optimistic protocols

Proper use of locks with multi lock modes

- Two types of locks: shared (for read only) and exclusive (for read and write, especially for write)

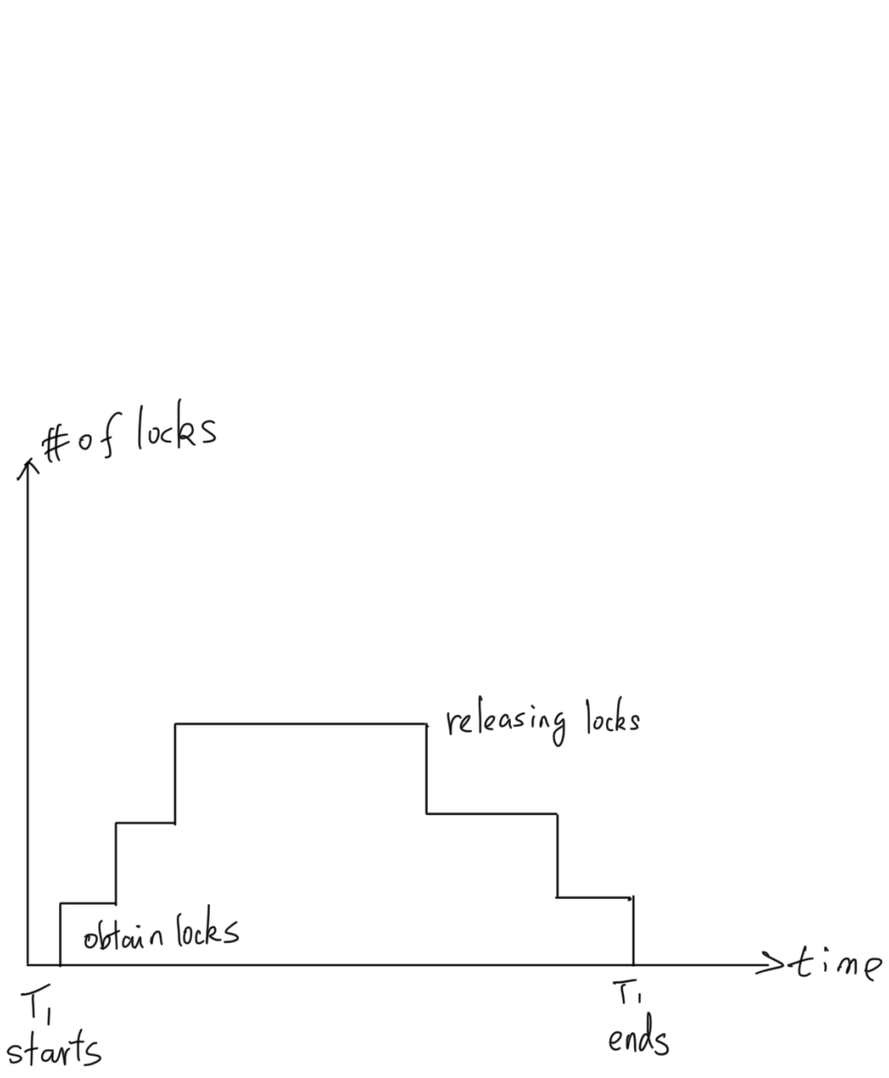
- $SL_i(x)$ means T_i requests a shared lock on element x.
  $XL_i(x)$ means T_i requests an exclusive lock on x.
  $U_i(x)$ means T_i releases whatever lock it has on x.

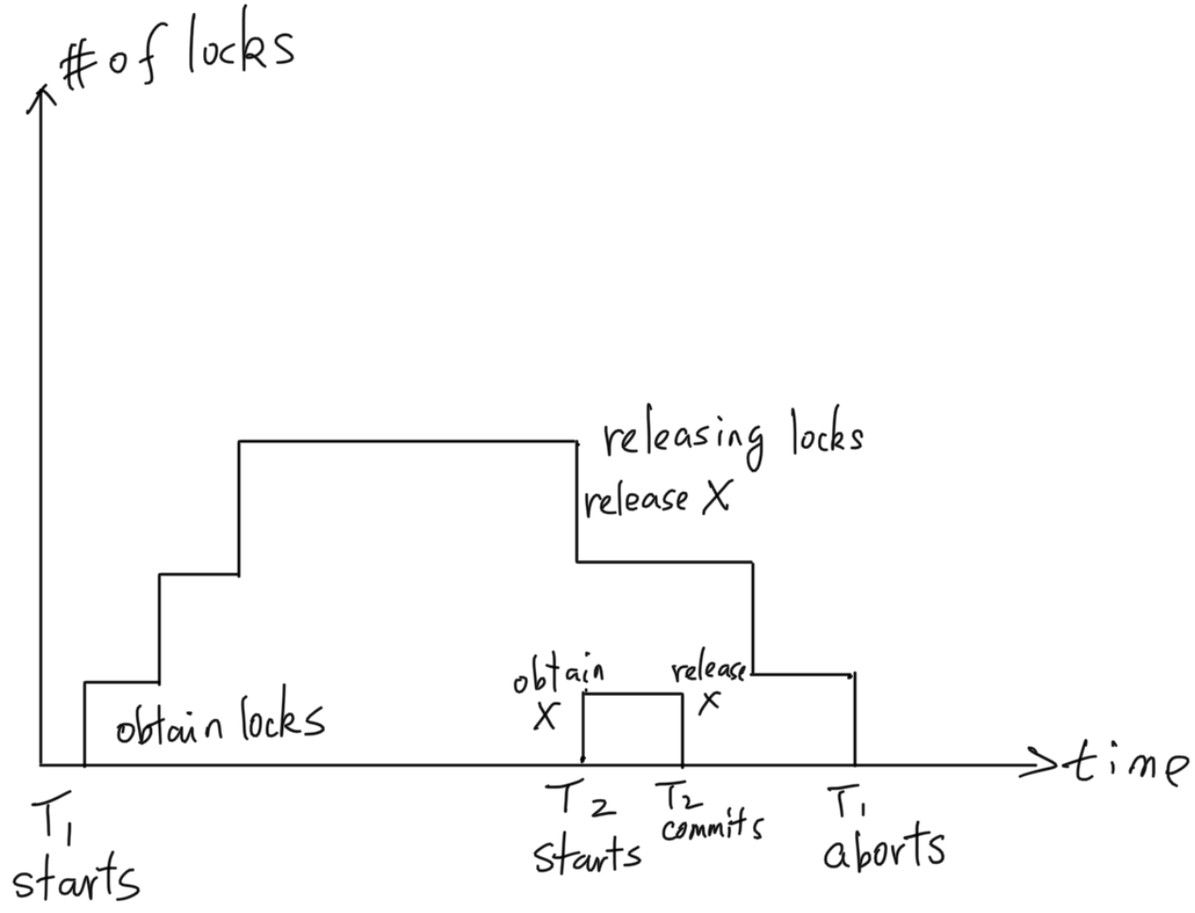
- Consistency of Transactions: Actions and locks must relate in the expected ways:
  - A transaction can only read or write an element if it previously was granted a proper lock on that element and has not yet released the lock.
  - If a transaction locks an element, it must later unlock that element.
- Legality of Schedules: Locks must have their intended meaning: An element may either be locked exclusively by one transaction or locked in shared mode by one or many transactions, but not both.

Two Phase Locking (2PL)

- In every transaction, all lock actions precede all unlock actions.
- Using 2PL, it is guaranteed to generate conflict serializable schedules. (But neither recoverable nor cascade-less schedule.)
- Why 2PL works? Each 2PL locked transaction may be thought to execute in its entirety at the instant it issues its first unlock request, so there is at least one conflict equivalent serial schedule: the one in which the transactions appear in the same order as their first unlock.
- Strict 2PL: all locks released at once when the transaction commits or aborts. Scheduler using strict 2PL can generate cascade-less schedules.

Visualization





Deadlock

- Could happen when lock based protocol is used by a scheduler.
- Solution:
  - Deadlock detection and recovery: Wait-for graph, cyclic graph means deadlock happened, then abort a transaction to break the circle.
  - Deadlock prevention: order the resource items, a transaction must request the items in a fixed order; or order the transactions and use "wait-die" or "wound-wait" protocols:

    T_1 request a lock held by T_2:

	wait-die	wound-wait
T_1 is older than T_2	T_1 waits	T_2 aborts
T_1 is younger than T_2	T_1 aborts	T_1 waits

Timestamp

- Each transaction has a ”timestamp”. TS(T)
- Each database object has following timestamps and an additional bit:
    - RT(X), read time of X, which is the highest timestamp of a transaction that has read X.
    - WT(X), write time of X, which is the highest timestamp of a transaction that has written X.
    - C(X), commit bit for X, is true if and only if the most recent transaction to write X has already committed. This is to avoid read the dirty data of a later aborted transaction.
- physically unrealizable action:
    - read too late, TS(T) < WT(X)
    - write too late, WT(X) < TS(T) < RT(X)

The Rules for Timestamp-Based Scheduling

- T requests to commit: find all elements X written by T and set C(X) = true. Release transactions waiting for T to commit.

- T requests to abort: any transaction waiting on an element X that T wrote must repeat its attempt to read or write X.

- T requests to read X

    - if TS(T) >= WT(X), it is realizable.
      if C(X) is true, execute it; set RT(X) = max(RT(X), TS(T);
      if C(X) is false, delay request until C(X) becomes true, or the transaction wrote X aborts.

    - if TS(T) < WT(X), it is physically unrealizable, abort T.

- T requests to write X

    - if TS(T) >= RT(X) and TS(T) >= WT(X), it is physically realizable, and will be executed:
      Update X; WT(X) = TS(T); C(X) = false;

    - if TS(T) >= RT(X) and TS(T) < WT(X), it is physically realizable, but a newer value is already in X, so
      if C(X) is true, ignore this request;
      if C(X) is false, delay request until C(X) becomes true or the transaction wrote X aborts.

    - if TS(T) < RT(X), it is physically unrealizable, abort T.

Optimistic Protocol

- Timestamp protocol is typically more aggressive than locking protocol.
- Locking will frequently delay transactions when waiting for locks.
- Timestamp introduces more delay if aborts happen frequently.
- Optimistic Protocol assumes that conflict rarely happens and proceeds each transaction in the following steps:
  - read: read all it needs, and compute all the values to be written and save them to its write set;
  - validate: make sure that its read and write set won't cause conflict with other ongoing transactions. If validate fails, this transaction aborts; otherwise, go on.
  - write: update database elements with the values in its write set.

Phantom Problem

T1	T2
select count(aid) from Accounts where ownerCid = 1234;	
	insert into Accounts values (a new account owned by 1234); update AccountStatistics set totalAcc = totalAcc+1 where ownerCid = 1234;
update AccountStatistics set totalAcc = (the count) where ownerCid = 1234;	

- Why the problem: T1 locked every tuple, but the new tuple did not exist, so it was un-lockable.

- Solution: treat the insertion and deletion as write operations on the whole table so they require exclusive locks to be performed.

SQL Isolation Levels

- you can tell DBMS how you would like your transaction be treated.

- set transaction [read only | read write] isolation level [read uncommitted | read committed | repeatable read | serializable];

Isolation Level	Dirty Reads	Non-Rep reads	Phantoms
read uncom	allowed	allowed	allowed
read committed	not allowed	allowed	allowed
repeatable read	not allowed	not allowed	allowed
serializable	not allowed	not allowed	not allowed