DBMS Concurrency Control

In a multiprogramming environment where more than one transactions can be concurrently executed, there exists a need of protocols to control the concurrency of transaction to ensure atomicity and isolation properties of transactions.

Concurrency control protocols, which ensure serializability of transactions, are most desirable. Concurrency control protocols can be broadly divided into two categories:

- Lock based protocols
- Time stamp based protocols

Lock based protocols

Database systems, which are equipped with lock-based protocols, use mechanism by which any transaction cannot read or write data until it acquires appropriate lock on it first. Locks are of two kinds:

- **Binary Locks:** a lock on data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive:** this type of locking mechanism differentiates lock based on their uses. If a lock is acquired on a data item to perform a write operation, it is exclusive lock. Because allowing more than one transactions to write on same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

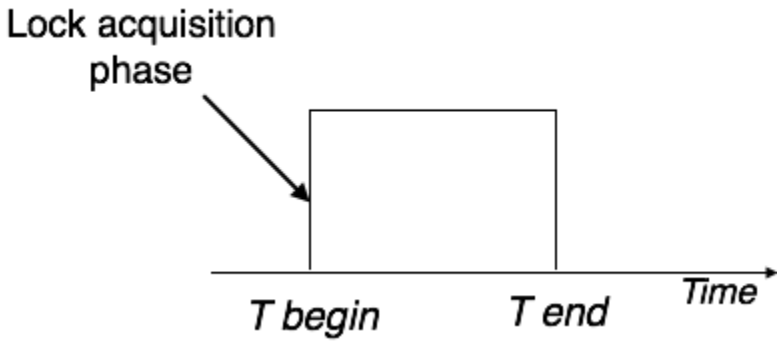
There are four types lock protocols available:

- **Simplistic**

  Simplistic lock based protocols allow transaction to obtain lock on every object before 'write' operation is performed. As soon as 'write' has been done, transactions may unlock the data item.

- **Pre-claiming**

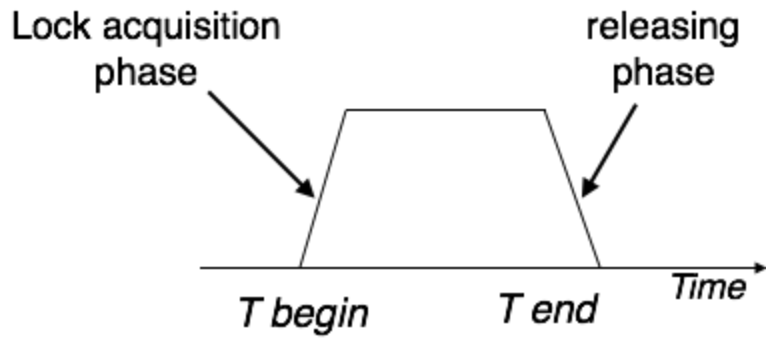
  In this protocol, a transactions evaluations its operations and creates a list of data items on which it needs locks. Before starting the execution, transaction requests the system for all locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. Else if all the locks are not granted, the transaction rolls back and waits until all locks are granted.



[*Image: Pre-claiming*]

- **Two Phase Locking - 2PL**

This locking protocol is divides transaction execution phase into three parts. In the first part, when transaction starts executing, transaction seeks grant for locks it needs as it executes. Second part is where the transaction acquires all locks and no other lock is required. Transaction keeps executing its operation. As soon as the transaction releases its first lock, the third phase starts. In this phase a transaction cannot demand for any lock but only releases the acquired locks.



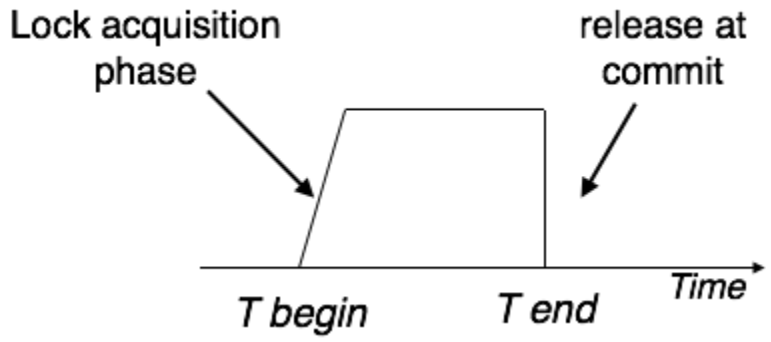
[*Image: Two Phase Locking*]

Two phase locking has two phases, one is growing; where all locks are being acquired by transaction and second one is shrinking, where locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to exclusive lock.

- **Strict Two Phase Locking**

  The first phase of Strict-2PL is same as 2PL. After acquiring all locks in the first phase, transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release lock as soon as it is no more required, but it holds all locks until commit state arrives. Strict-2PL releases all locks at once at commit point.



[*Image: Strict Two Phase Locking*]

Strict-2PL does not have cascading abort as 2PL does.

Time stamp based protocols

The most commonly used concurrency protocol is time-stamp based protocol. This protocol uses either system time or logical counter to be used as a time-stamp.

Lock based protocols manage the order between conflicting pairs among transaction at the time of execution whereas time-stamp based protocols start working as soon as transaction is created.

Every transaction has a time-stamp associated with it and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transaction, which come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and priority may be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know, when was last read and write operation made on the data item.

TIME-STAMP ORDERING PROTOCOL

The timestamp-ordering protocol ensures serializability among transaction in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- Time-stamp of Transaction Ti is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows:

- **If a transaction Ti issues read(X) operation:**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  - All data-item Timestamps updated.

- **If a transaction Ti issues write(X) operation:**
  - If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

THOMAS' WRITE RULE:

This rule states that in case of:

- If TS(Ti) < W-timestamp(X)
-
- Operation rejected and Ti rolled back. Timestamp ordering rules can be modified to make the schedule view serializable. Instead of making Ti rolled back, the 'write' operation itself is ignored.

Source:

http://www.tutorialspoint.com/dbms/dbms_concurrency_control.htm