**(Locking Techniques)**

**Implementation of Locking in DBMS**

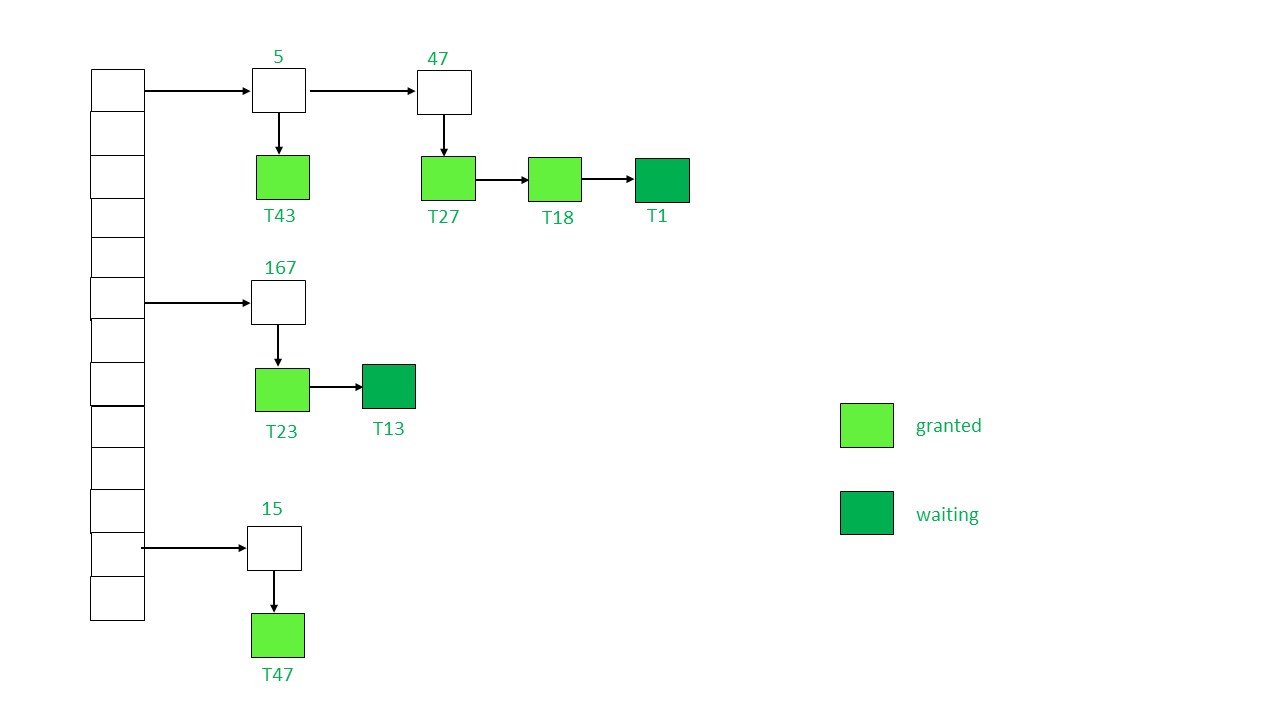
Locking protocols are used in database management systems as a means of concurrency control. Multiple transactions may request a lock on a data item simultaneously. Hence, we require a mechanism to manage the locking requests made by transactions. Such a mechanism is called as **Lock Manager**. It relies on the process of message passing where transactions and lock manager exchange messages to handle the locking and unlocking of data items.

**Data structure used in Lock Manager –**

The data structure required for implementation of locking is called as **Lock table**.

1. It is a hash table where name of data items are used as hashing index.
2. Each locked data item has a linked list associated with it.
3. Every node in the linked list represents the transaction which requested for lock, mode of lock requested (mutual/exclusive) and current status of the request (granted/waiting).
4. Every new lock request for the data item will be added in the end of linked list as a new node.
5. Collisions in hash table are handled by technique of separate chaining.

Consider the following example of lock table:



**Explanation:** In the above figure, the locked data items present in lock table are 5, 47, 167 and 15.

The transactions which have requested for lock have been represented by a linked list shown below them using a downward arrow.

Each node in linked list has the name of transaction which has requested the data item like T33, T1, T27 etc.

The colour of node represents the status i.e. whether lock has been granted or waiting.

Note that a collision has occurred for data item 5 and 47. It has been resolved by separate chaining where each data item belongs to a linked list. The data item is acting as header for linked list containing the locking request.

**Working of Lock Manager –**

1. Initially the lock table is empty as no data item is locked.
2. Whenever lock manager receives a lock request from a transaction Ti on a particular data item Qi following cases may arise:

* If Qi is not already locked, a linked list will be created and lock will be granted to the requesting transaction Ti.
* If the data item is already locked, a new node will be added at the end of its linked list containing the information about request made by Ti.

1. If the lock mode requested by Ti is compatible with lock mode of transaction currently having the lock, Ti will acquire the lock too and status will be changed to ‘granted’. Else, status of Ti’s lock will be ‘waiting’.
2. If a transaction Ti wants to unlock the data item it is currently holding, it will send an unlock request to the lock manager. The lock manager will delete Ti’s node from this linked list. Lock will be granted to the next transaction in the list.
3. Sometimes transaction Ti may have to be aborted. In such a case all the waiting request made by Ti will be deleted from the linked lists present in lock table. Once abortion is complete, locks held by Ti will also be released.

**Lock-Based Protocol**

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

**1. Shared lock:**

* It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
* It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

**2. Exclusive lock:**

* In the exclusive lock, the data item can be both reads as well as written by the transaction.
* This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

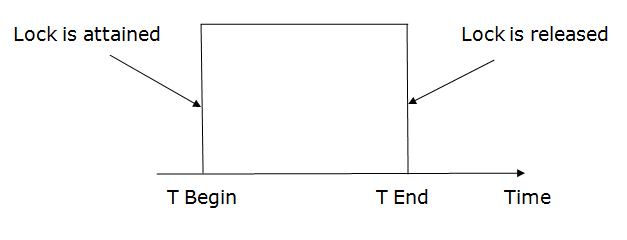
**There are four types of lock protocols available:**

**1. Simplistic lock protocol**

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

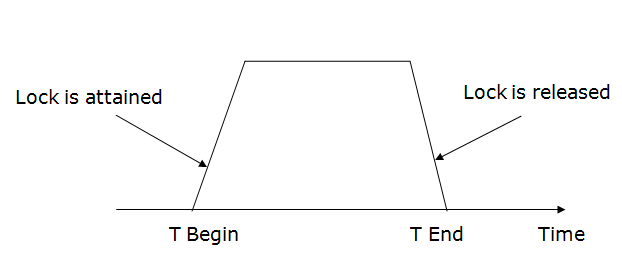
**2. Pre-claiming Lock Protocol**

* Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
* Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
* If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
* If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



**3. Two-phase locking (2PL)**

* The two-phase locking protocol divides the execution phase of the transaction into three parts.
* In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
* In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
* In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

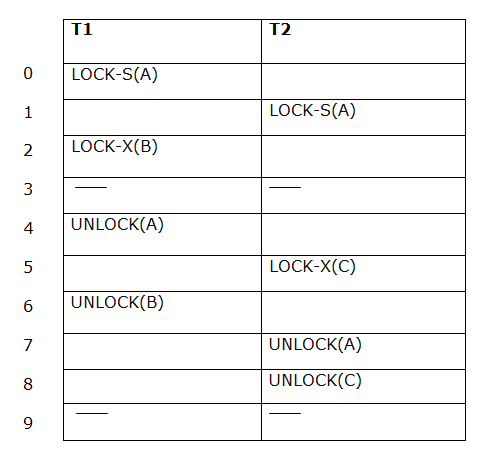
**Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

**Shrinking phase:** In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

**Example:**



The following way shows how unlocking and locking work with 2-PL.

**Transaction T1:**

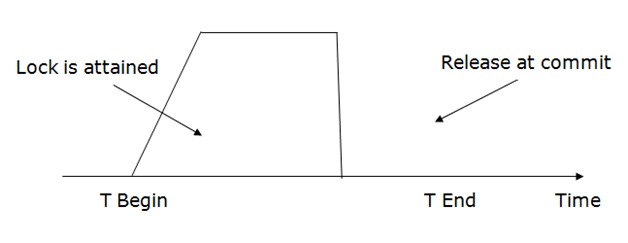
* **Growing phase:** from step 1-3
* **Shrinking phase:** from step 5-7
* **Lock point:** at 3

**Transaction T2:**

* **Growing phase:** from step 2-6
* **Shrinking phase:** from step 8-9
* **Lock point:** at 6

**4. Strict Two-phase locking (Strict-2PL)**

* The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
* The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
* Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
* Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

**What is Lock? Type of Lock in DBMS.**

Transaction processing systems usually allow multiple transactions to run concurrently. By allowing multiple transactions to run concurrently will improve the performance of the system in terms of increased throughout or improved response time, but this allows causes several complications with consistency of the data. Ensuring consistency in spite of concurrent execution of transaction require extra work, which is performed by the concurrency controller system of DBMS.

**What is Lock?**

A lock is a variable associated with a data item that describes the status of the item withrespect to possible operations that can be applied to it. Generally, there is one lock for each dataitem in the database. Locks are used as a means of synchronizing the access by concurrenttransactions to the database item.

**Types of Locks**

Several types of locks are used in concurrency control. To introduce locking concepts gradually, we first discuss binary locks, which are simple but restrictive and so are not used in practice. We then discuss shared/exclusive locks, which provide more general locking capabilities and are used in practical database locking schemes.

**Binary Locks**

A binary lock can have two states or values: locked and unlocked.

A distinct lock is associated with each database item *A.*If the value of the lock on *A*is 1, item *A*cannot be accessed by a database operation that requests the item. If the value of the lock on *A*is 0 then item can be accessed when requested. We refer to the current value of the lock associated with item *A*as *LOCK (A).*There are two operations, lock item and unlock item are used with binary locking A transaction requests access to an item *A*by first issuing a lock*item (A)*operation. If LOCK (A) = 1, the transaction is forced to wait. If LOCK (A) = 0 it is set to 1 (the transaction locks the item) and the transaction is allowed to access item *A.*When the transaction is through using the item, it issues an unlock *item (A)*operation, which sets *LOCK (A)* to 0 (unlocks the item) so that *A*may be accessed by other transactions. Hence binary lock enforces mutual exclusiol1 on the data item.

**Rules of Binary Locks**

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

1. A transaction must issue the operation lock\_item (A) before any read\_item (A) or write, item operations are performed in T.

2. A transaction T must issue the operation unlock\_item (A) after all read\_item (A) and write\_item (A) operations are completed in T.

3. A transaction T will not issue a lock *\_item (A)*operation if it already holds the lock on Item *A.*

4. A transaction T will not issue an unlock *\_item (A)*operation unless it already holds the lock on item *A.*

5.  The lock manager module of the DBMS can enforce these rules. Between the Lock*\_item (A)*and unlock\_*item (A)*operations in transaction T, is said to hold the lock on item *A.*At most one transaction can hold the lock on a particular item. Thus no two transactions can access the’ same item concurrently.

**Disadvantages of Binary Locks**

As discussed earlier, binary locking scheme is too restrictive for database items, because at most one transaction can hold a lock on a given item. So, binary locking system cannot be used for practical purpose.

**Share/Exclusive (for Read/Write) Locks**

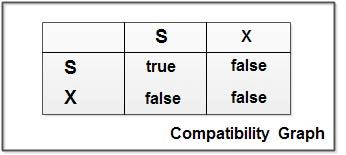
We should allow several transactions to access the same item *A*if they all access *A’*for reading purposes only. However, if a transaction is to write an item A, it must have exclusive access to *A.*For this purpose, a different type of lock called a multiple-mode lock is used. In this scheme there are shared/exclusive or read/write locks are used.

**Locking operations**

There are three locking operations called read\_lock(A), write\_lock(A) and unlock(A) represented as lock-S(A), lock-X(A), unlock(A) (Here, S indicates shared lock, X indicates exclusive lock)can be performed on a data item. A lock associated with an item A, LOCK (A), now has three possible states: “read-locked”, “write-locked,” or “unlocked.” A read-locked item is also called share-locked item because other transactions are allowed to read the item, whereas a write-locked item is caused exclusive-locked, because a single transaction exclusively holds the lock on the item.

**Compatibility of Locks**

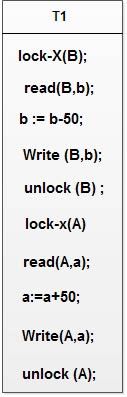
Suppose that there are A and B two different locking modes. If a transaction T1 requests a lock of mode on item Q on which transaction T2 currently hold a lock of mode B. If transaction can be granted lock, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a function is shown in one matrix as shown below:



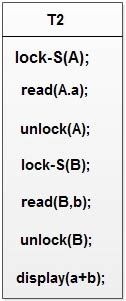
The graphs shows that if two transactions only read the same data object they do not conf1ict, but if one transaction writes a data object and another either read or write the same data object, then they conflict with each other. A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction. Similarly, an exclusive lock is requested through the lock- X(Q) instruction. A data item Q can be unlocked via the unlock(Q) instruction.

To access a data item, transaction T1 must first lock that item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. Thus, T1 is made to wait until all incompatible locks held by other transactions have been released

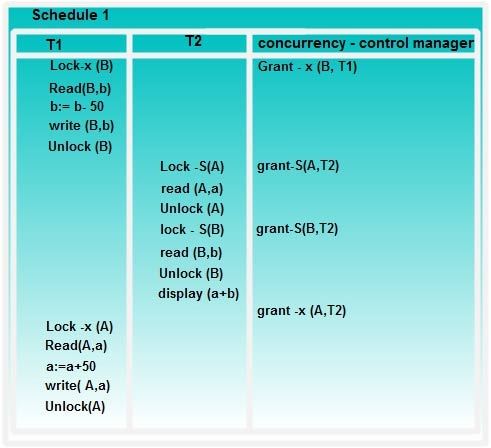
Example: As an illustration consider the simplified banking system. Let A and B be two accounts that are accessed by transactions T1 and T2. Transaction T1 transfers Rs.50 from account B to account a and is defined as:



Transaction T2 displays the total amount of money in accounts A and B that is, the sum A+B and is defined as

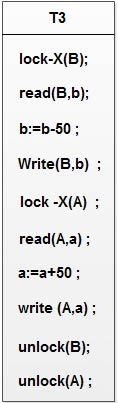


Suppose that the values of accounts A and Bare Rs.100 and Rs.200, respectively. If these two transactions are executed serially, either in the order T1 and T2 or the order T2, T1 then transaction T2 will display the value Rs.300. If however, these transactions are executed concurrently, as shown in Schedule 1. In this case, transaction T2 displays Rs.250, which is incorrect. The reason for this mistake is that the transaction TI unlocked data item B too early, as a result of which T2 shows an inconsistent state.

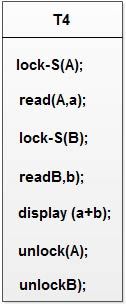
****

**Solution of Inconsistency Problem**

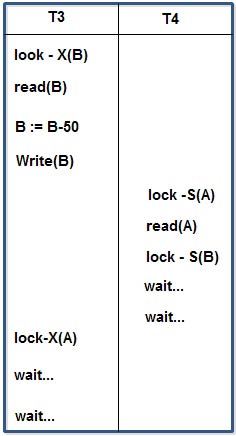
Suppose now that unlocking is delayed to the end of the transaction. The transaction T3 corresponds to T I with unlocking delayed and is defined as



**Transaction T4 corresponds to T2 with unlocking delayed, and is defined as**

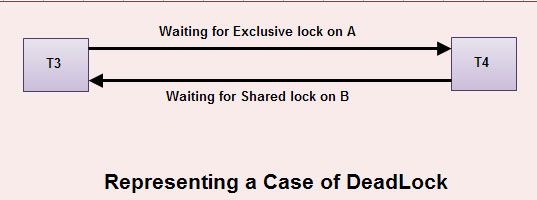


You should verify that the sequence of reads and writes in schedule I which leads to an incorrect total of Rs.250 being displayed, is no longer possible with T3 and T4 as shown in Schedule 2.

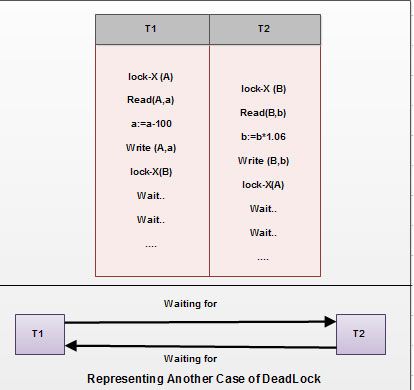


**Schedule 2**

Unfortunately, the use of locking can lead to an undesirable situation. Consider the partial schedule 2, T3 is holding an exclusive mode lock on B and T4 is requesting a shared mode lock on B i.e.T4 is waiting forT3 to unlock B. Similarly, T4 is holding a shared mode lock on A and T3 is requesting an exclusive mode lock on A, thus T3 is waiting for T4 to unlock A.



Thus we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This situation is called deadlock.



**Conclusion**

Thus we can say that the solution of inconsistency leads to deadlock problem. If we do not use locking or unlock data items as soon as possible after reading or writing them, we may get inconsistent states. On the other hand, if we do not unlock a data item before requesting a lock on another data item deadlocks may occur. There are ways to avoid deadlock in some situations. Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back of transactions, where as inconsistent states may lead to real world problems that cannot be handled by the database system.

**OTHER REFRENCES**

•      [What is Lock? Type of Lock in DBMS. - Computer Notes (ecomputernotes.com)](https://ecomputernotes.com/database-system/rdbms/type-of-lock-in-dbms)

•      [Implementation of Locking in DBMS - GeeksforGeeks](https://www.geeksforgeeks.org/implementation-of-locking-in-dbms/)

•      [DBMS Lock based Protocol - javatpoint](https://www.javatpoint.com/dbms-lock-based-protocol)

**SUGGESTED BOOK REFERENCES**

* C.J.Date, “An Introduction to DatabaseSystems”, Addison Wesley.
* Thomas M. Connolly, Carolyn & E.Begg,“Database Systems: A Practical Approach to Design, Implementationand Management”, 5/E, University of Paisley, Addison-Wesley.
* Rob,”Database Principal Fundamental Design, Cengage Learning.