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

BY DINESH THAKUR

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](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-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**

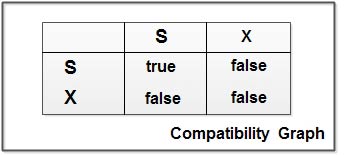
We should allow several transactions to access the same item *A*if they all access *A'*for reading purposes only. However, if a transaction is to write an item A, it must have exclusive access to *A.*For this purpose, a different type of lock called a multiple-mode lock is used. In this scheme there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

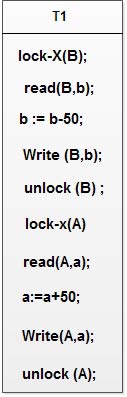
Suppose that there are A and B two different locking modes. If a transaction T1 requests a lock of mode on item Q on which transaction T2 currently hold a lock of mode B. If transaction can be granted lock, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such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

[](http://ecomputernotes.com/images/compatibility-graph.jpg)

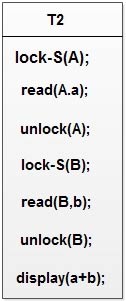
The graphs shows that if two transactions only read the same data object they do not conf1ict, but if one transaction writes a data object and another either read or write the same data object, then they conflict with each other. A transaction requests a shared lock on data item Q by executing the lock-S(Q) 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 1'1 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

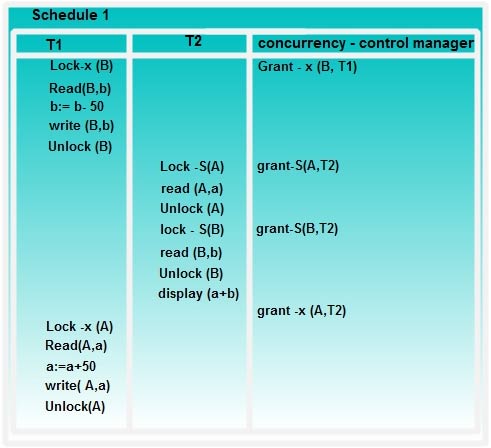
Example: As an illustration consider the simplified banking system. Let A and B be two accounts that are accessed by transactions TI and T2. Transaction TI transfers Rs.50 from account B to account a and is defined as:

[](http://ecomputernotes.com/images/T1%20con.jpg)

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

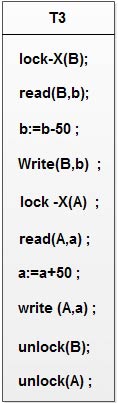
[](http://ecomputernotes.com/images/T2%20con.jpg)

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 T 1 and T2 or the order T2, T 1 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.

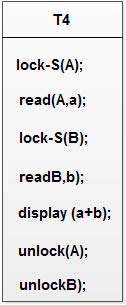
[](http://ecomputernotes.com/images/Schedule.jpg)

**Solution of Inconsistency Problem**

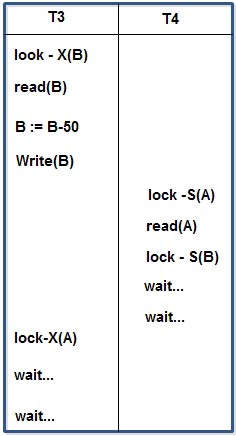
Suppose now that unlocking is delayed to the end of the transaction. The transaction T3 corresponds to T I with unlocking delayed and is defined as

[](http://ecomputernotes.com/images/T3%20con.jpg)

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

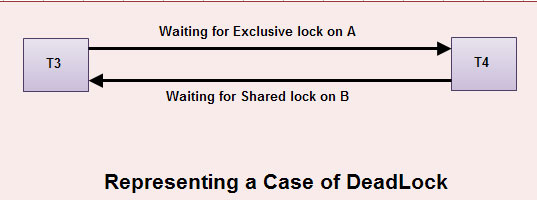
[](http://ecomputernotes.com/images/T4%20con.jpg)

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.

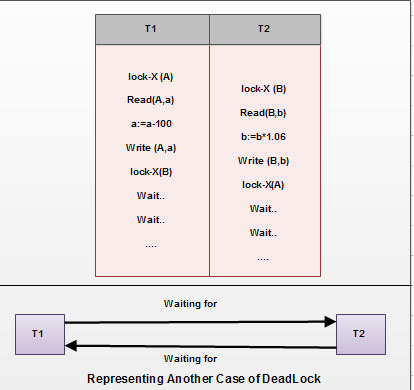
[](http://ecomputernotes.com/images/Schedule%202.jpg)

**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.

[](http://ecomputernotes.com/images/representing-a-case-of-deadlock.jpg)

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

**[](http://ecomputernotes.com/images/representing-another-case-of-deadlock.jpg)**

**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.