Two Phase Locking Protocols and Deadlock

The Two-Phase Locking (2PL) Protocol is an essential concept in database management systems used to maintain data consistency and ensure smooth operation when multiple transactions are happening simultaneously. It helps to prevent issues like data conflicts where two or more transactions try to access or modify the same data at the same time, potentially causing errors.

Two-Phase Locking is widely used to ensure serializability, meaning transactions occur in a sequence that maintains data accuracy. This article will explore the workings of the 2PL protocol, its types, advantages and its role in maintaining a reliable database system **Two Phase Locking**

The Two-Phase Locking (2PL) Protocol is a key technique used in database management systems to manage how multiple transactions access and modify data at the same time. When many users or processes interact with a database, it's important to ensure that data remains consistent and error-free. Without proper management, issues like data conflicts or corruption can occur if two transactions try to use the same data simultaneously.

The Two-Phase Locking Protocol resolves this issue by defining clear rules for managing data locks. It divides a transaction into two phases:

1. **Growing Phase:** In this step, the transaction gathers all the locks it needs to access the required data. During this phase, it cannot release any locks.
2. **Shrinking Phase:** Once a transaction starts releasing locks, it cannot acquire any new ones. This ensures that no other transaction interferes with the ongoing process.

Types of Lock

Shared Lock (S): Shared Lock is also called a read-only lock, allows multiple transactions to access the same data item for reading at the same time. However, transactions with this lock cannot make changes to the data. A shared lock is requested using the lock-S instruction.

Exclusive Lock (X): An Exclusive Lock allows a transaction to both read and modify a data item. This lock is exclusive, meaning no other transaction can access the same data item while this lock is held. An exclusive lock is requested using the lock-X instruction.

Lock Conversions

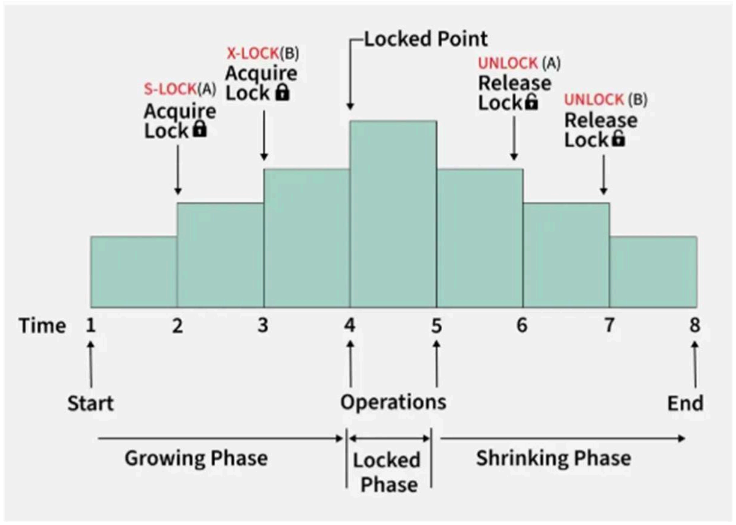
In the Two-Phase Locking Protocol, lock conversion means changing the type of lock on data while a transaction is happening. This process is carefully controlled to maintain consistency in the database.

1. **Upgrading a Lock:** This means changing a shared lock (S) to an exclusive lock (X). For

example, if a transaction initially only needs to read data (S) but later decides it needs to update the same data, it can request an upgrade to an exclusive lock (X). However, this can only happen during the Growing Phase, where the transaction is still acquiring locks.

- Example: A transaction reads a value (S lock) but then realizes it needs to modify the value. It upgrades to an X lock during the Growing Phase.

2. **Downgrading a Lock:** This means changing an exclusive lock (X) to a shared lock (S). For instance, if a transaction initially planned to modify data (X lock) but later decides it only needs to read it, it can downgrade the lock. However, this must happen during the Shrinking Phase, where the transaction is releasing locks.
   - Example: A transaction modifies a value (X lock) but later only needs to read the value, so it downgrades to an S lock during the Shrinking Phase.



2PL-Locking

These rules ensure that the Two-Phase Locking Protocol maintains consistency and avoids conflicts between transactions. By limiting when upgrades and downgrades can occur, the system prevents situations where multiple transactions interfere with each other's operations.

Let’s see a transaction implementing 2-PL.

	T1	T2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	……….	……….
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	……….	……….

This is a basic outline of a transaction that demonstrates how locking and unlocking work in the Two-Phase Locking Protocol (2PL).

Transaction T1

- The growing Phase is from steps 1-3
- The shrinking Phase is from steps 5-7
- Lock Point at

3 Transaction T2

- The growing Phase is from steps 2-6
- The shrinking Phase is from steps 8-9
- Lock Point at 6

Lock Point

The lock point in a transaction is the moment when the transaction finishes acquiring all the locks it needs. After this point, no new locks can be added, and the transaction starts releasing locks. It’s a key step in the Two-Phase Locking Protocol to ensure the rules of growing and shrinking phases are followed.

Example of 2PL

Imagine a library system where multiple users can borrow or return books. Each action (like borrowing or returning) is treated as a transaction. Here's how the Two-Phase Locking Protocol (2PL) works, including the lock point:

User A wants to:

1. Check the availability of Book X.
2. Borrow Book X if it's available.
3. Update the library's record.

Growing Phase (Locks are Acquired):

1. User A locks Book X with a shared lock (S) to check its availability.
2. After confirming the book is available, User A upgrades the lock to an exclusive lock (X) to borrow it.
3. User A locks the library's record to update the borrowing details.

   Lock Point: Once User A has acquired all the necessary locks (on Book X and the library record), the transaction reaches the lock point. No more locks can be acquired after this.

Shrinking Phase (Locks are Released):

1. User A updates the record and releases the lock on the library's record.
2. User A finishes borrowing and releases the exclusive lock on Book X.

   This process ensures that no other user can interfere with Book X or the library record during the transaction, maintaining data accuracy and consistency. The lock point ensures that all locks are acquired before any are released, following the 2PL rules.

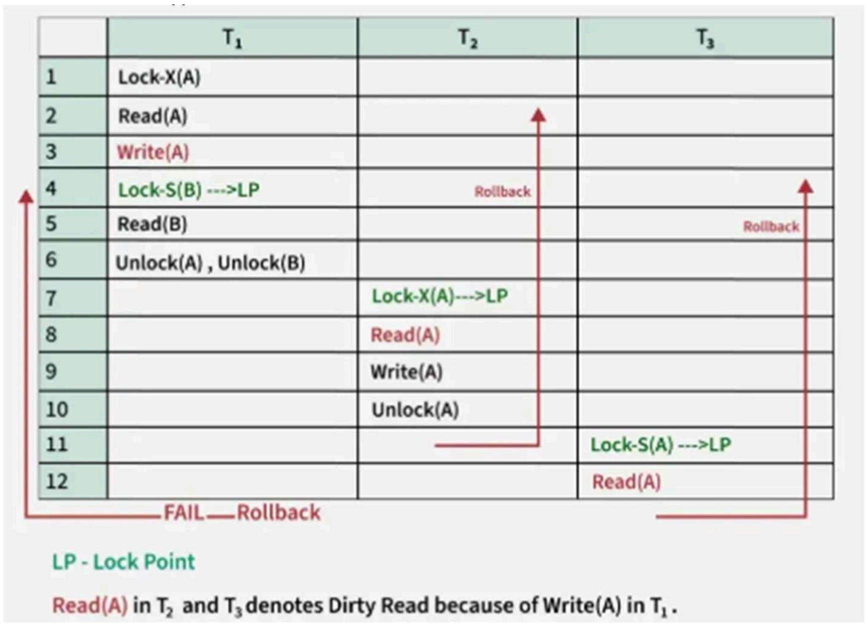
Drawbacks of 2PL

Two-phase locking (2PL) ensures that transactions are executed in the correct order by using two phases: acquiring and releasing locks. However, it has some drawbacks:

- **Deadlocks:** Transactions can get stuck waiting for each other's locks, causing them to freeze indefinitely.
- **Cascading Rollbacks:** If one transaction fails, others that depend on it might also fail leading to inefficiency and potential data issues.
- **Lock Contention:** Too many transactions competing for the same locks can slow down the system, especially when many users are working at the same time.
- **Limited Concurrency:** The strict rules of 2PL can reduce how many transactions can run at once, resulting in slower performance and longer wait times.

Cascading Rollbacks in 2-PL

Let’s see the following Schedule:



	T_1	T_2	T_3
1	Lock-X(A)		
2	Read(A)		
3	Write(A)		
4	Lock-S(B) --->LP		
5	Read(B)		
6	Unlock(A) , Unlock(B)		
7		Lock-X(A)--->LP	
8		Read(A)	
9		Write(A)	
10		Unlock(A)	
11			Lock-S(A) --->LP
12			Read(A)

Rollback
Rollback
FAIL—Rollback

LP - Lock Point

Read(A) in T_2 and T_3 denotes Dirty Read because of Write(A) in T_1 .

Schedule

The image illustrates a transaction schedule using the Two-Phase Locking (2PL) protocol, showing the sequence of actions for three transactions T1, T2 and T3.

Key Points:

1. Transaction T1:
    - T1 acquires an exclusive lock (X) on data item A, performs a write operation on A and then acquires a shared lock (S) on B.
    - T1 reaches its lock point (LP) after acquiring all locks.
    - Eventually, T1 fails and a rollback is triggered, undoing its changes.
2. Transaction T2:
    - T2 reads A after T1 writes A. This is called a dirty read because T1’s write is not committed yet.
    - When T1 rolls back, T2’s operations become invalid, and it is also forced to rollback.
3. Transaction T3:
    - T3 reads A after T2 reads A. Since T2 depends on the uncommitted changes of T1, T3 indirectly relies on T1’s changes.
    - When T1 rolls back, T3 is also forced to rollback even though it was not directly interacting with T1’s operations.
    - Cascading Rollback Problem:
    - Here, T2 and T3 both depend on uncommitted data from T1 (either directly or indirectly).

- When T1 fails and rolls back all dependent transactions (T2 and T3) must also rollback because their operations were based on invalid data.
- Cascading rollbacks waste system resources, reduce concurrency and lead to inefficiency.
- In large systems, this can significantly affect performance and make recovery more complex.

How to avoid it?

- Strict 2PL: Ensure that transactions release their locks only after they commit, preventing other transactions from accessing uncommitted changes.
- Rigorous 2PL: Extend strict 2PL to hold both read and write locks until the transaction commits, offering even stronger guarantees.

Deadlock in 2-PL

Consider this simple example. We have two transactions T1 and T2.

Schedule: Lock-X1(A) Lock-X2(B) Lock-X1(B) Lock-X2(A)

This sequence represents a locking scenario where two transactions, T1 and T2 are trying to lock two resources, A and B in a particular order. Here's what each step means:

1. Lock-X1(A):
   Transaction T1 acquires an exclusive lock on resource A. This means T1 has full control over A and no other transaction can use it until T1 releases the lock.
2. Lock-X2(B):
   Transaction T2 acquires an exclusive lock on resource B. Similarly, T2 now has full control over B and no other transaction can access B until T2 releases the lock.
3. Lock-X1(B):
   Transaction T1 tries to acquire an exclusive lock on resource B but T2 already holds the lock on B. So, T1 must wait for T2 to release the lock.
4. Lock-X2(A):
   At the same time, Transaction T2 tries to acquire an exclusive lock on resource A but T1 already holds the lock on A. So, T2 must wait for T1 to release the lock.

The above-mentioned type of 2-PL is called Basic 2PL. To sum it up, it ensures Conflict Serializability but does not prevent Cascading Rollback and Deadlock. To prevent from these issues Strict Two-Phase Locking and Rigorous Two-Phase Locking can be used.