Concurrency Control

Dr. Seema Gupta Bhol

DBMS Concurrency Control

- Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database.

- The simultaneous execution should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database.

Advantages of Concurrency

- **Less Waiting Time:** If a process is in a ready state but still the process does not get the system to get execute is called waiting time.
- **Less Response Time:** The time wasted in getting the response from the CPU for the first time, is called response time. So, concurrency leads to less Response Time.
- **Efficient Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to more Resource Utilization.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to more Efficiency.

Concurrency Control Problems

Following five problems occur with the Concurrent Execution of the operations:

- Lost Update Problem
- Dirty Read /Temporary Update Problem
- Unrepeatable Read Problem
- Incorrect Summary Problem
- Phantom Read Problem

Lost Update Problems

- The problem occurs when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.

- When two or more transactions try to update the same data item at the same time, a lost update happens, and the outcome relies on the sequence in which the transactions are executed. The modifications made by the other transaction will be lost if one transaction overwrites them before they are committed.

For example: Consider the following diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.
- Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problem

- The dirty read problem occurs when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.

- When a transaction accesses data that has already been updated but hasn't been committed, it's known as a dirty read. The information read by the first transaction will be invalid if the modifying transaction rolls back.

For example: Consider two transactions T_X and T_Y in the following diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

Dirty Read Problem

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem

- Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

- When a transaction reads the same data item twice and the data is updated by another transaction between the two reads, this is known as a non-repeatable read. This might result in discrepancies in outcomes and data.

For example:

- Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300.

Unrepeatable Read Problem

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

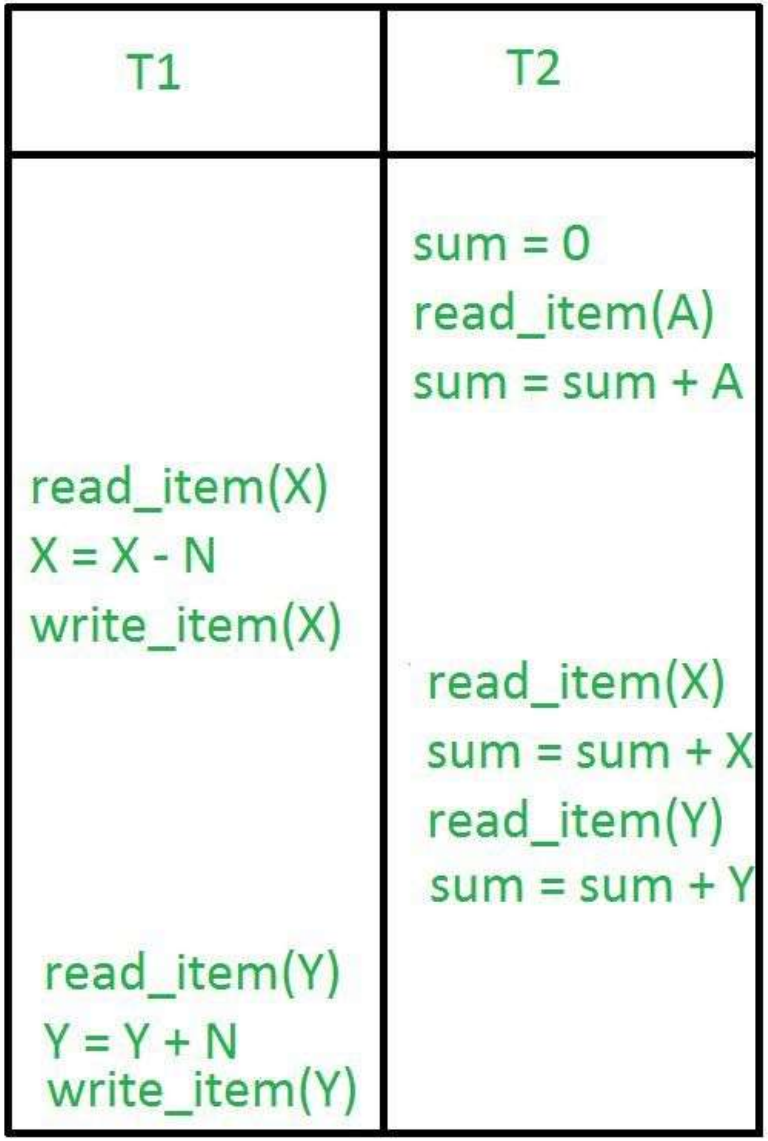
Within the same transaction T_X, it reads two different values of account A

Unrepeatable Read Problem (W-R Conflict)

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Incorrect Summary Problem

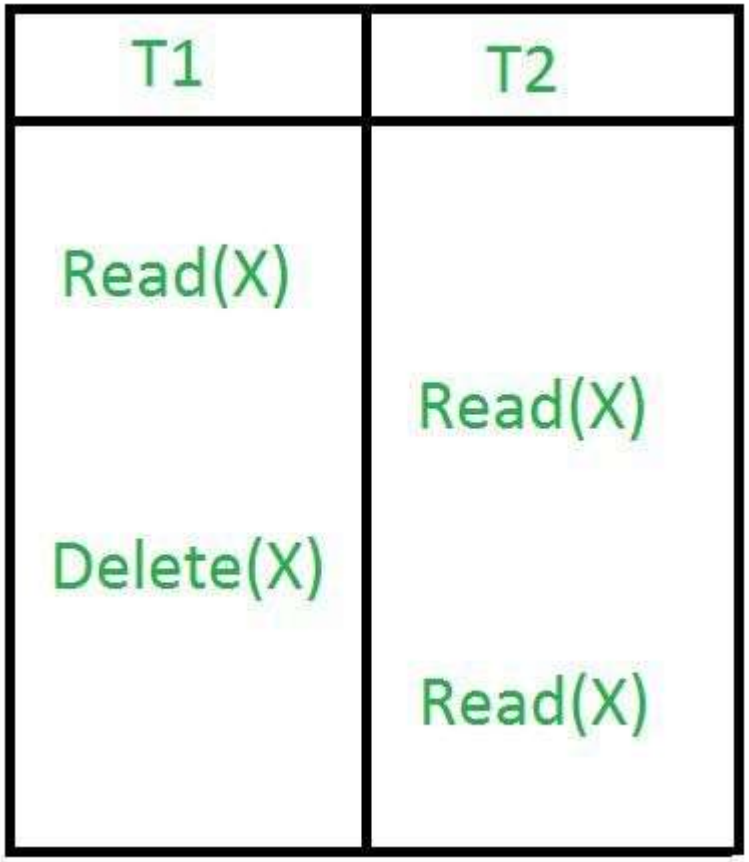
Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.



Here, transaction 2 is calculating the sum of some records while transaction 1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.

Phantom Read Problem

The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.



Locking protocols

- 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 a **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.

Lock-Based Protocols

- Data items can be locked in two modes :

  1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.

  2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to the concurrency-control manager by the programmer. Transaction can proceed only after request is granted.

Locks

- A lock is a mechanism to control concurrent access to a data item
- A lock is a variable associated with a data item that describes a status of data item with respect to possible operation that can be applied to it.
- Two common locks which are used :
- Shared Lock (S): also known as Read-only lock.
- Exclusive Lock (X): Data item can be both read as well as written.

Shared Locks

- 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.
- S-lock is requested using lock-S instruction.
- Consider the situation where the value of variable X equals 50 and there are a total of 2 transactions reading X. If one transaction wants to change the value of X, another transaction that tries to read the value will read the incorrect value of the variable X. However, until it is done with reading, the Shared lock stops it from updating.

Exclusive Lock (X)

- This type of lock is Exclusive and cannot be held simultaneously on the same data item.
- X-lock is requested using lock-X instruction.
- After finishing the 'write' step, transactions can unlock the data item.
- Example of exclusive locks: Consider the instance where the value of a data item X is equal to 50 and a transaction requires a deduction of 20 from the data item X. By putting a Y lock on this particular transaction, we can make it possible. As a result, the exclusive lock prevents any other transaction from reading or writing.

Shared and Exclusive locks

Shared Lock

- A transaction T_1 -having been shared can only read the data.
- More than one transaction can acquire a shared lock on the same variable.
- Represented by S

Exclusive Lock

- A transaction T_1 -having an exclusive lock can both read as well as write
- At any given time, only one Transaction can have an exclusive lock on a variable.
- Represented by X

Lock-Based Protocols

- **Lock-compatibility matrix**

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
  - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Implementation of Locking

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked

Example 1

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

Time	Tx	Ty
t1	Lock-Xread(A)	-
t2	A=A-50	-
t3	-	Lock-XRead (A)
t4	-	wait
t5	-	wait
t6	Write (A)	wait
t7	Unlock (A)	
		A=A+100
		Write(A)

Example 2

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

Time	Tx	Ty
t1	Lock-Xread(A)	-
t2	A=A+50	-
t3	Write (A)	-
t4	-	Lock-XRead(A)
t5	rollback	-
t6	lock on A released	
t7		Lock granted
		-
		-

Example-3

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

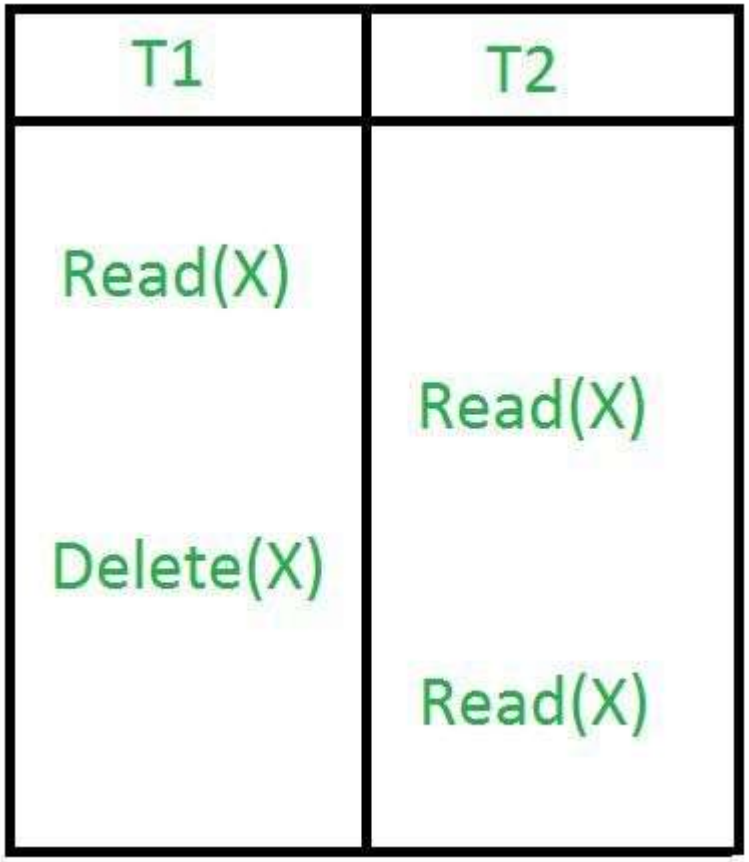
Time	Tx	Ty
t1	Lock-Xread(A)	-
t2	-	Lock-XRead(A)
t3	-	wait
t4	-	wait
t5	Read(A)	wait
t6	-	wait

Example 4

T1	T2
	sum = 0 read_item(A) sum = sum + A
read_item(X) X = X - N write_item(X)	
	read_item(X) sum = sum + X read_item(Y) sum = sum + Y
read_item(Y) Y = Y + N write_item(Y)	

Time	T1	T2
t1	-	Sum=0
t2	--	Lock-XRead(A)
t3	-	Sum=sum+A
t4	Lock-XRead(X)	-
t5	X=X-N	-
t6	Write(x)	-
t7	Unlock X	Lock-XRead(X)
		Sum=Sum+x
		Lock-XRead(y)
		Sum=sum+Y
	Lock-XRead (y)	Unlock y
	Y=Y + n	
	Write (y)	
	Unlock Y	

Example 5



Time	T1	T2
t1	Lock-Xread(X)	-
t2		Lock-XRead(A)
t3		wait
t4	Delete(X)	wait
t5		Lock not granted

Conversion between locks

The two methods outlined below can be used to convert between the locks:

- Conversion from a read lock (s) to a write lock(x) is an upgrade.
- Conversion from a write lock (x) to a read lock(s) is a downgrade.

Issues with Locks

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

T1
lock-X(B);
read(B,b);
b := b-50;
Write (B,b);
unlock (B) ;
lock-x(A)
read(A,a);
a:=a+50;
Write(A,a);
unlock (A);

T2
lock-S(A);
read(A.a);
unlock(A);
lock-S(B);
read(B,b);
unlock(B);
display(a+b);

Schedule 1

T1	T2	concurrency - control manager
Lock-x (B)		Grant - x (B, T1)
Read(B,b)		
b:= b- 50		
write (B,b)		
Unlock (B)		
	Lock -S(A)	grant-S(A,T2)
	read (A,a)	
	Unlock (A)	
	lock - S(B)	grant-S(B,T2)
	read (B,b)	
	Unlock (B)	
	display (a+b)	
		grant -x (A,T2)
Lock -x (A)		
Read(A,a)		
a:=a+50		
write(A,a)		
Unlock(A)		

Suppose that the values of accounts A and B are 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 Problem

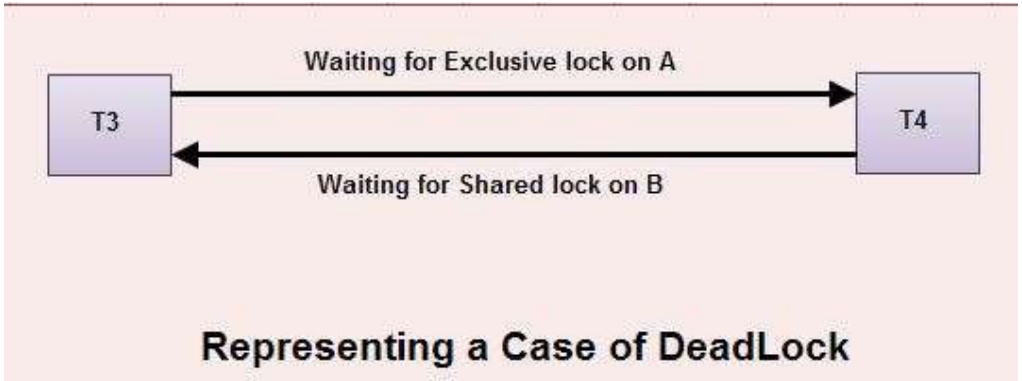
- Suppose now that unlocking is delayed to the end of the transaction. The transaction T3 corresponds to T I with unlocking delayed .Transaction T4 corresponds to T2 with unlocking delayed.

T3
lock-X(B);
read(B,b);
b:=b-50 ;
Write(B,b) ;
lock -X(A) ;
read(A,a) ;
a:=a+50 ;
write (A,a) ;
unlock(B);
unlock(A) ;

T4
lock-S(A);
read(A,a);
lock-S(B);
readB,b);
display (a+b);
unlock(A);
unlockB);

T3	T4
look - X(B)	
read(B)	
B := B-50	
Write(B)	
	lock -S(A)
	read(A)
	lock - S(B)
	wait...
	wait...
lock-X(A)	
wait...	
wait...	

- 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.
- Unfortunately, the use of locking can lead to an undesirable situation.
- 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.



Representing a Case of DeadLock

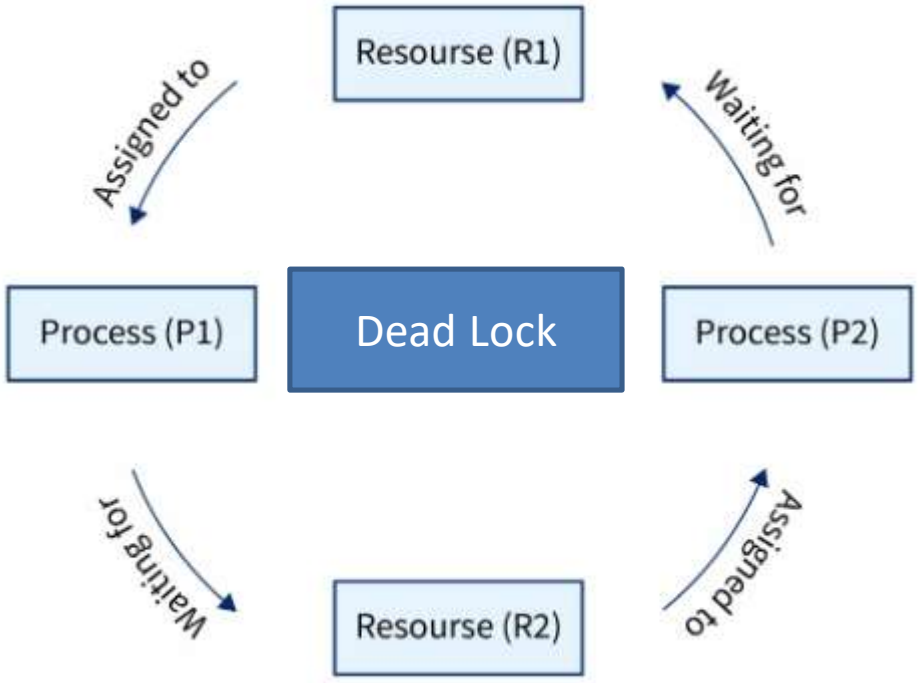
- Thus we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This situation is called deadlock.

Starvation and Deadlock

- When a transaction must wait an unlimited period for a lock, it is referred to as starvation. The following are the causes of starvation
  - When the locked item waiting scheme is not correctly controlled.
  - When a resource leak occurs.
  - The same transaction is repeatedly chosen as a victim.
- The resource allocation priority scheme should contain ideas like aging, in which a proces's priority rises as it waits longer. This prevents starvation.

Deadlock

A deadlock situation occurs when two or more processes are expecting each other to release a resource or when more than 2 processes are waiting for the resource.



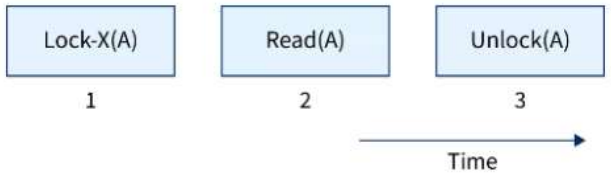
Types of Lock-Based Protocols

There are four lock based protocols in DBMS :

- Simplistic Lock Protocol
- Pre-claiming Lock Protocol
- Two-phase Locking Protocol
- Strict Two-Phase Locking Protocol.

Simplistic Lock Protocol

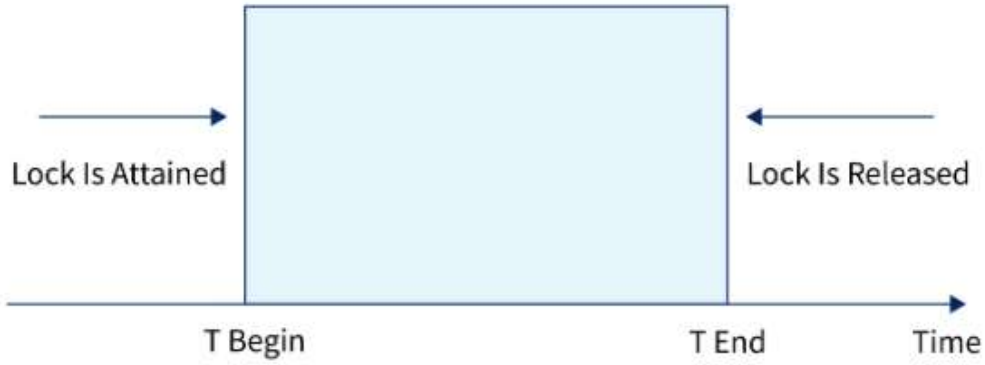
- The simplistic method is defined as the most fundamental method of securing data during a transaction.
- Simple lock-based protocols allow all transactions to lock the data before inserting, deleting, or updating it.
- The transaction releases the lock as soon as it is done performing the operation. This prevents other transactions to read the data while its being updated.



- This type of lock protocol may lack some of the advanced features and optimizations found in more complex protocols but still ensures basic data integrity.

Pre-Claiming Lock Protocol

- Pre-claiming Lock Protocols are known to assess transactions to determine which data elements require locks.
- Prior to actually starting the transaction, it asks the Database management system for all of the locks on all of the data items. The pre-claiming protocol permits the transaction to commence if all of the locks are obtained.



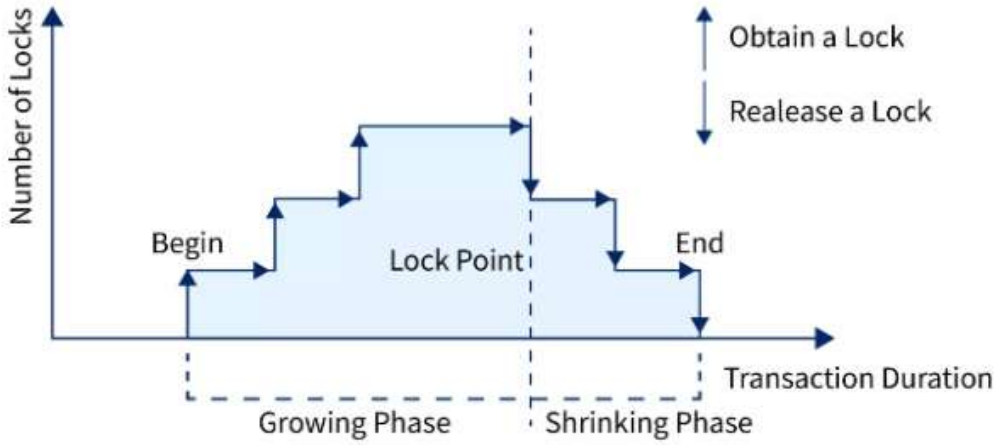
- Whenever the transaction is finished, the lock is released.
- This protocol permits the transaction to roll back if all of the locks are not granted and then waits until all of the locks are granted.

Pre-Claiming Lock Protocol

- This protocol aims to improve concurrency and reduce waiting times by allowing transactions to pre-claim locks before actually needing them.
- By pre-claiming locks, transactions reduce the likelihood of conflicts with other transactions that may require the same data items later.

Two-phase Locking Protocol

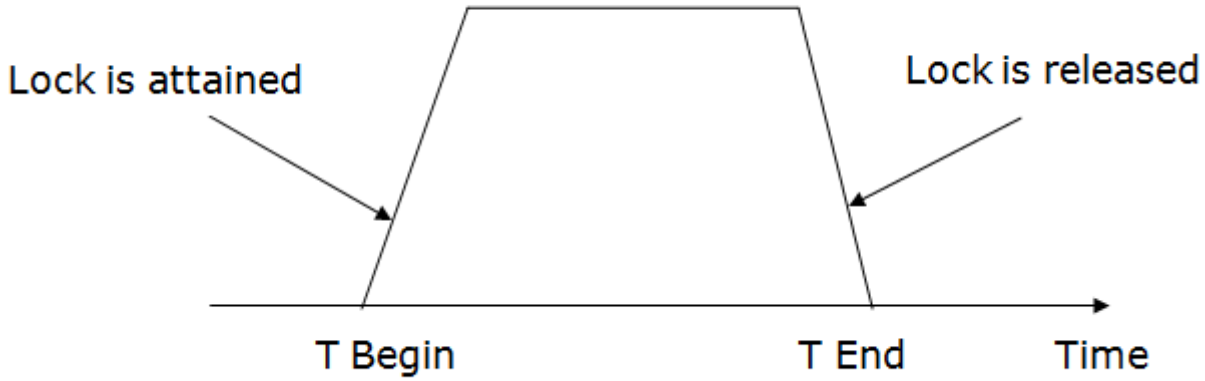
- If Locking as well as the Unlocking can be performed in 2 phases, a transaction is considered to follow the Two-Phase Locking protocol. The two phases are known as the growing and shrinking phases.



- Growing Phase: In this phase, we can acquire new locks on data items, but none of these locks can be released.
- Shrinking Phase: In this phase, the existing locks can be released, but no new locks can be obtained.

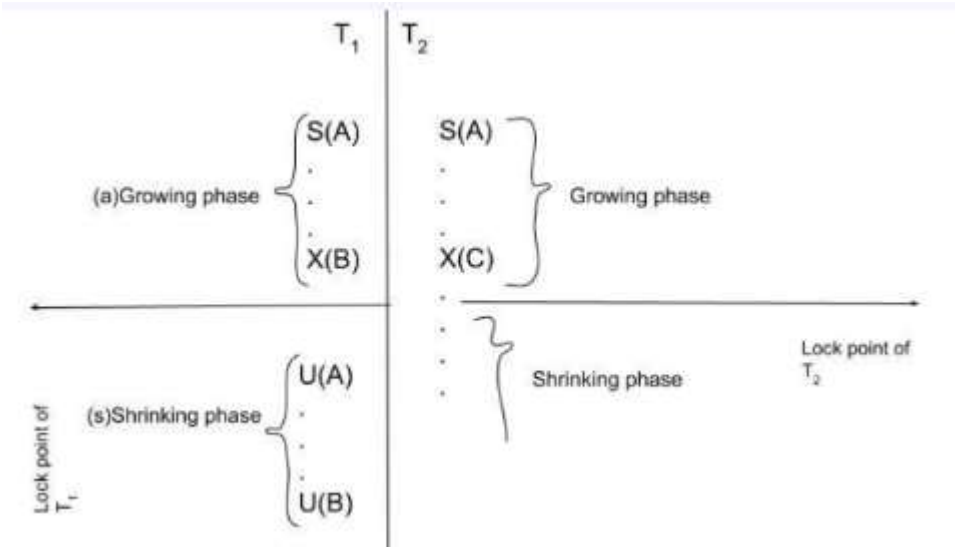
Two-phase Locking Protocol

A transaction is said to follow the **two-phase locking protocol** if *all* locking operations precede the *first* unlock operation in the transaction.



If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.

Growing phase, Lock point, Shrinking phase



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	…….	…….

Lock Point

The Point at which the growing phase ends, i.e., when a transaction takes the final lock it needs to carry on its work.

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

Guaranteeing Serializability by Two-Phase Locking

- If *every* transaction in a schedule follows the two-phase locking protocol, the schedule is *guaranteed to be serializable*.
- It ensures that the transactions are serializable conflict. The serializable conflict means when there is no cycle between the read and write operations of different transactions.
- 2PL guarantees conflict serializability, but the problems are cascading aborts, deadlock, and unnecessary or early acquiring of locks.

Example

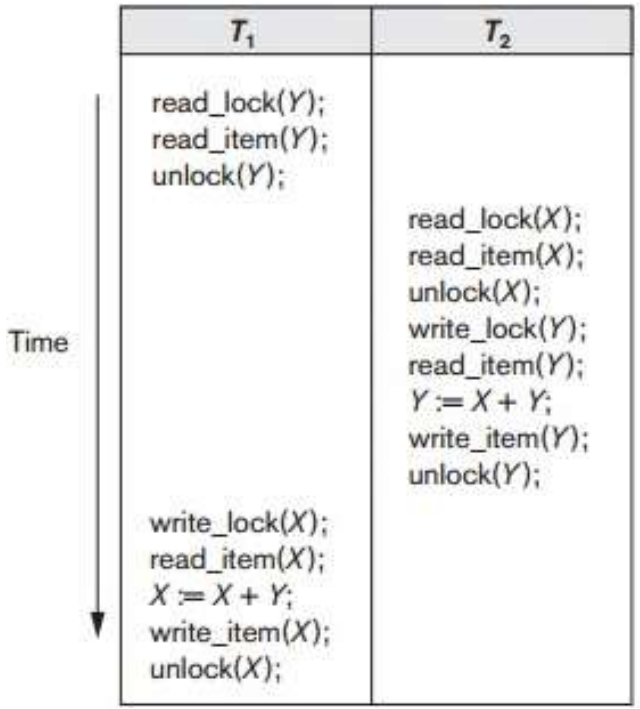
T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

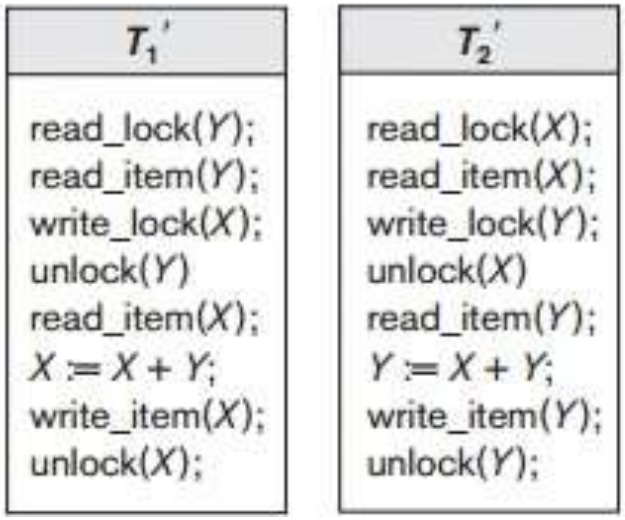
Result of serial schedule T_2 followed by T_1: X=70, Y=50



Result of schedule
X=50, Y=50
(nonserializable)

In the above schedule, transactions T_1 and T_2 do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T_1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T_2.

If we enforce two-phase locking, the transactions can be rewritten as T'_1 and T'_2



- It can be proved that, if *every* transaction in a schedule follows the two-phase locking protocol, the schedule is *guaranteed to be serializable*. However, there are still some drawbacks of 2-PL.
- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation are possible.

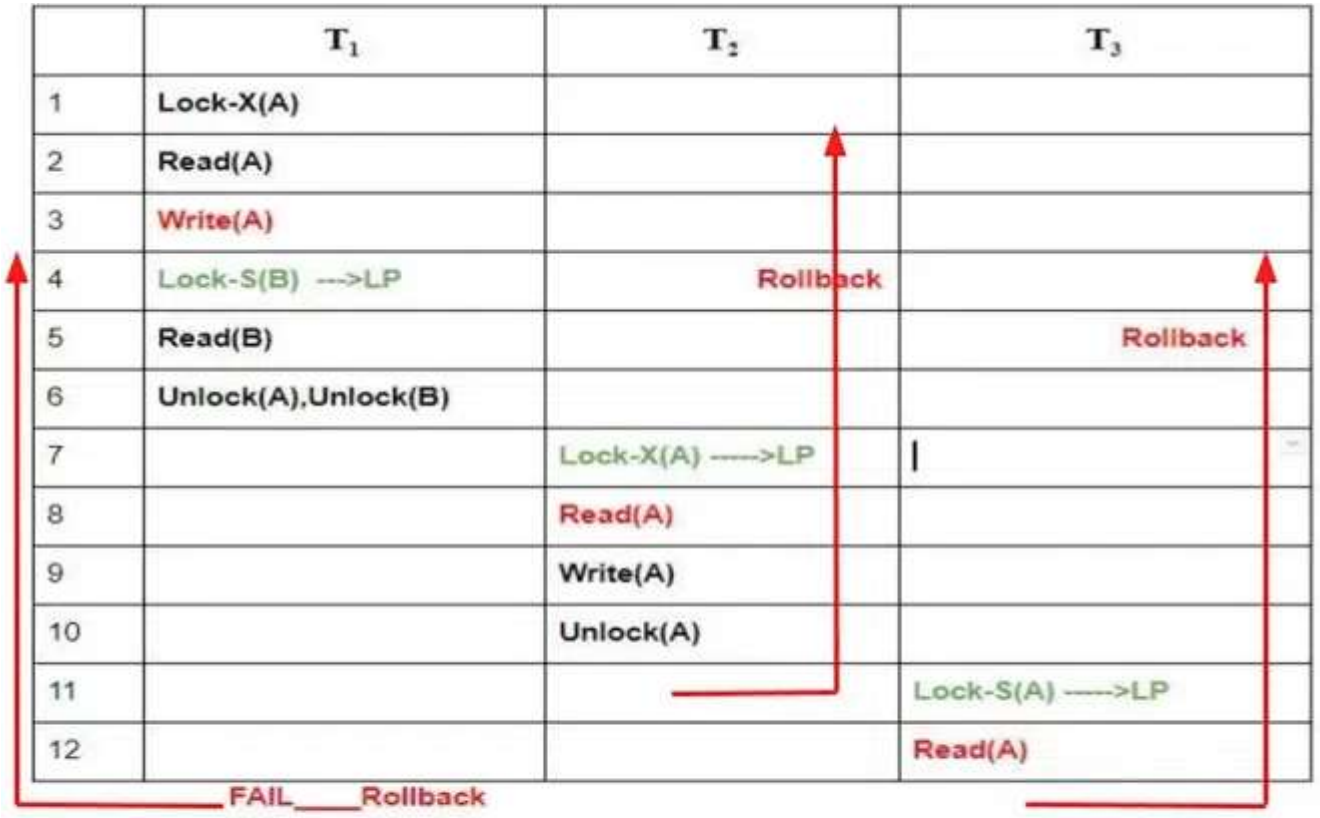
Limitations of Two-phase Locking Protocol

- Two-phase locking reduces the amount of concurrency in a schedule.
- The protocol raises transaction processing costs and may have unintended consequences.
- The likelihood of establishing deadlocks is one bad result.
- Cascading Rollback is possible under 2-PL.

Dead Lock with two phase Locking Protocol

Time	T'_1	T'_2	Lock Status
1	Lock-S (Y)	Lock-S (X)	Granted
2	Read(Y)	Read(X)	
3	Lock-X(X)	Lock-X(Y)	wait
4	wait	wait	
	----	---	
	---	----	

Cascading Rollbacks in 2-PL



LP - Lock Point

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

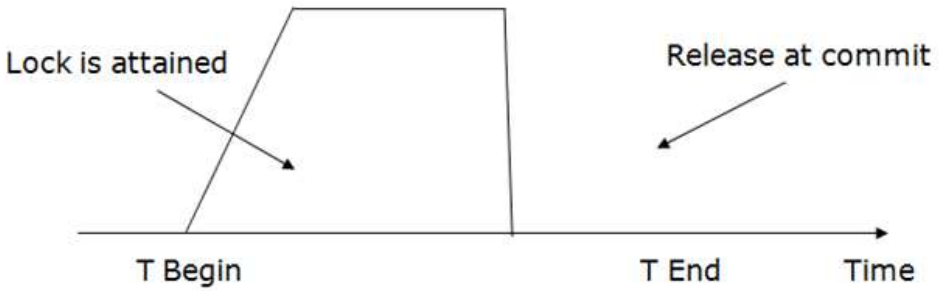
Because of Dirty Read in T_2 and T_3 in lines 8 and 12 respectively, when T_1 failed we have to roll back others also. Hence, Cascading Rollbacks are possible in 2-PL.

Strict Two-Phase Locking Protocol

- Cascaded rollbacks are avoided with the concept of a Strict Two-Phase Locking Protocol. This protocol necessitates not only two-phase locking but also the retention of all exclusive locks until the transaction commits or aborts.
- It is responsible for assuring that if one transaction modifies data, there can be no other transaction that will be able to read it until the first transaction commits.
- The majority of database systems use a strict two-phase locking protocol.

Strict Two-Phase Locking Protocol

- 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.