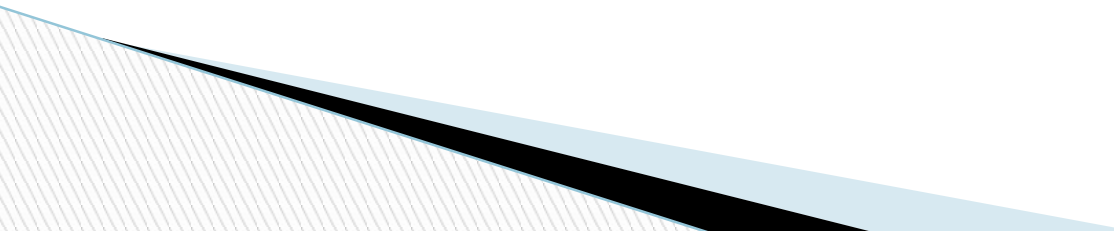


UNIT IV

Transaction Management and Query Processing

A decorative graphic at the bottom of the slide consisting of several overlapping, wavy, horizontal bands. From top to bottom, the bands are light blue, black, dark grey, and light grey with a fine diagonal line pattern.

Transaction

- A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.
 - Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.
- 

A's Account

Open_Account(A)

Old_Balance = A.balance

New_Balance = Old_Balance – 500

A.balance = New_Balance

Close_Account(A)

B's Account

Open_Account(B)

Old_Balance = B.balance

New_Balance = Old_Balance + 500

B.balance = New_Balance

Close_Account(B)



Operations of Transaction:

- ▣ **Read(X):** Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
- ▣ **Write(X):** Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. $R(X);$
2. $X = X - 500;$
3. $W(X);$

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:



- ❑ **Commit:** It is used to save the work done permanently.
- ❑ **Rollback:** It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction (**ACID PROPERTIES**)

Atomicity

Consistency

Isolation

Durability



Atomicity

means either all successful or none.

Consistency

ensures bringing the database from one consistent state to another consistent state.
ensures bringing the database from one consistent state to another consistent state.

Isolation

ensures that transaction is isolated from other transaction.

Durability

means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- ❖ It states that all operations of the transaction take place at once if not, the transaction is aborted.
- ❖ There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.



- Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:=A-100 Write(A)	Read(B) Y:=Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

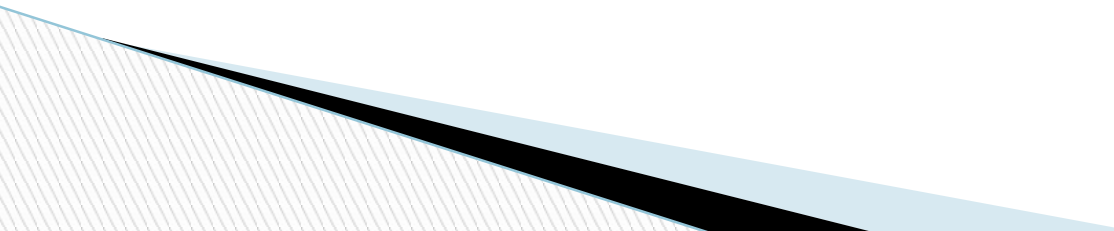
For example: The total amount must be maintained before or after the transaction.

Total before T occurs = $600+300=900$

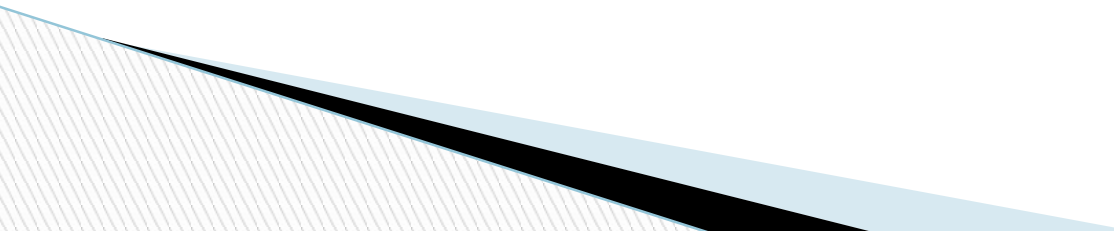
Total after T occurs = $500+400=900$

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

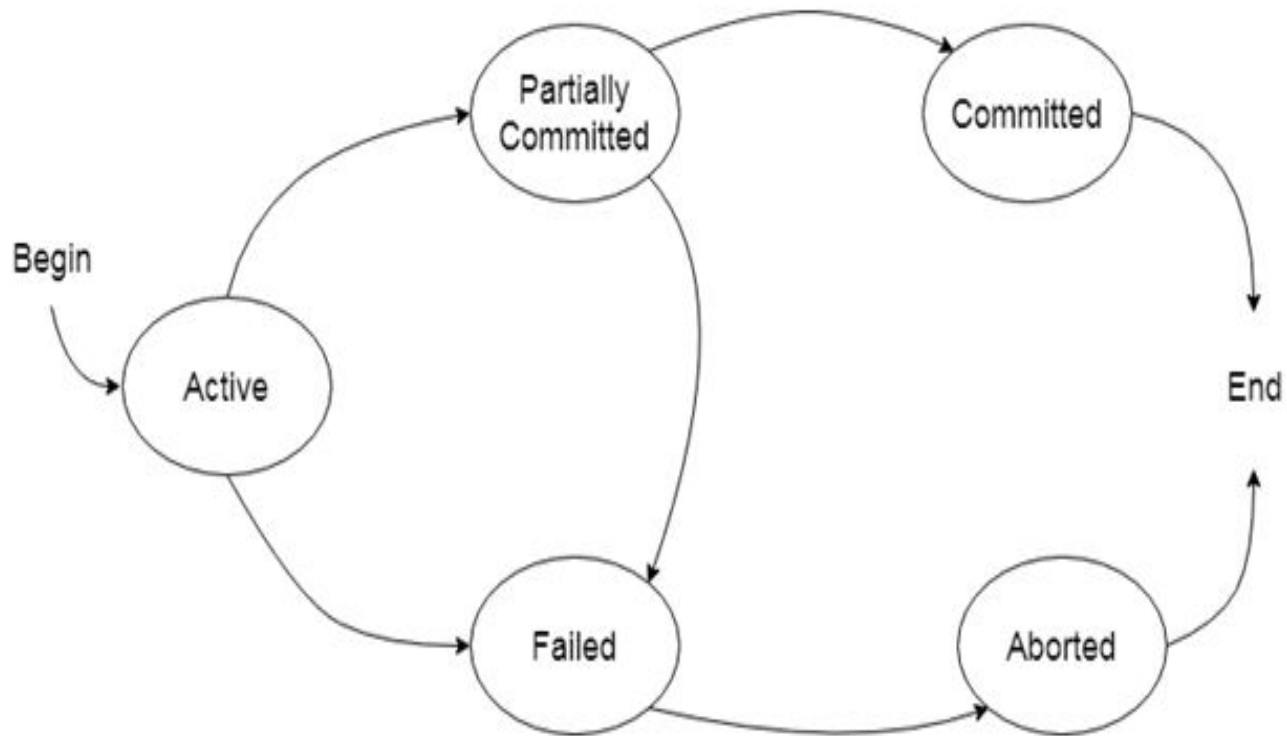
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
 - In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
 - The concurrency control subsystem of the DBMS enforced the isolation property.
- 

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
 - They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
 - The recovery subsystem of the DBMS has the responsibility of Durability property.
- 

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

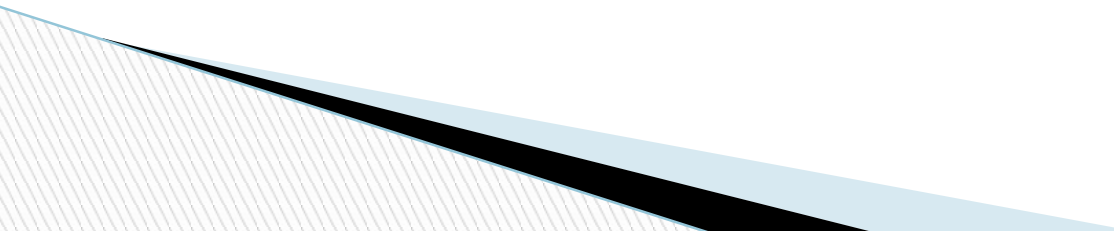
Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

- A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
 - In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.
- 

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
 - Re-start the transaction
 - Kill the transaction

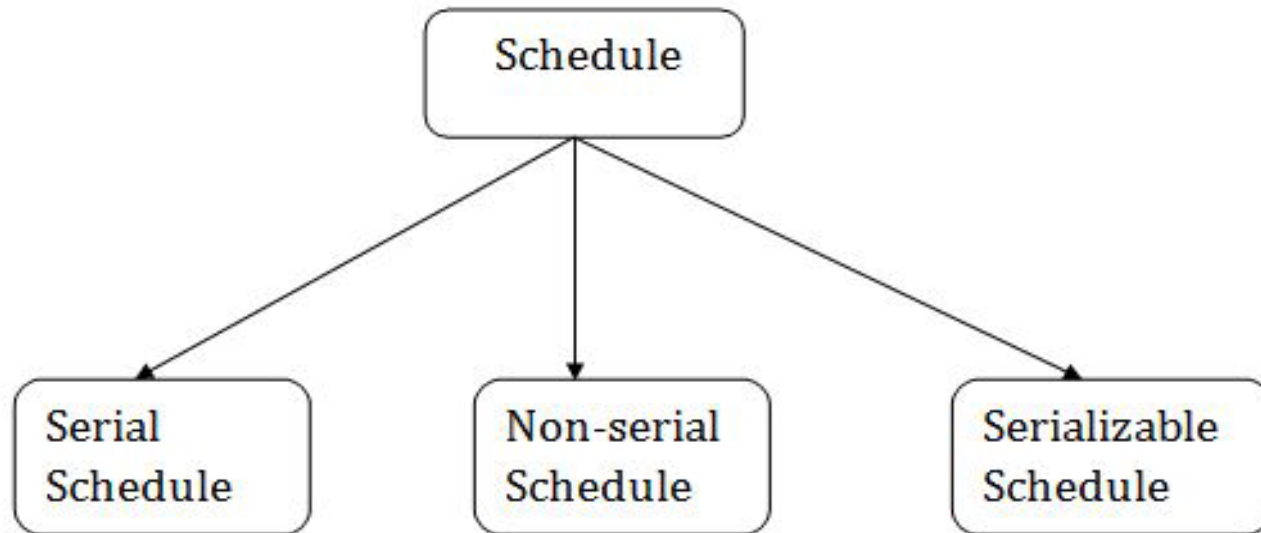
□



Schedule

A series of operation from one transaction to another transaction is known as schedule.

It is used to preserve the order of the operation in each of the individual transaction.



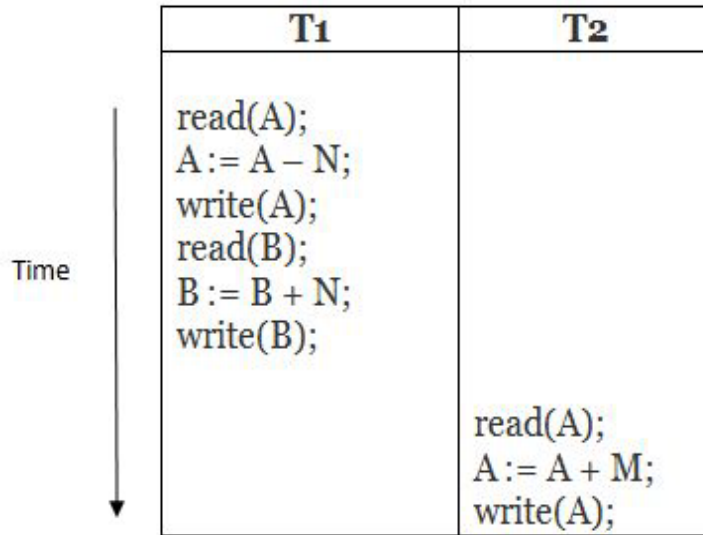
1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

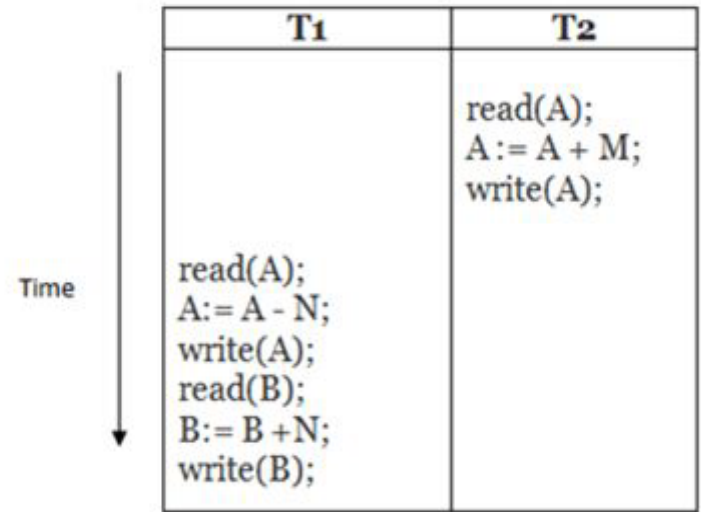
- Execute all the operations of T1 which was followed by all the operations of T2.
- Execute all the operations of T2 which was followed by all the operations of T1.

(a)



Schedule A

(b)



Schedule B

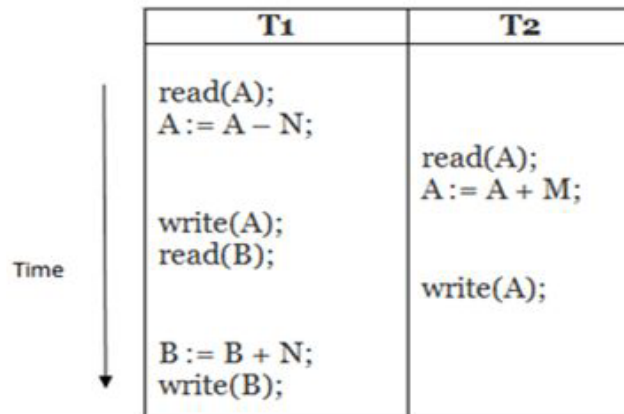
In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.

In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

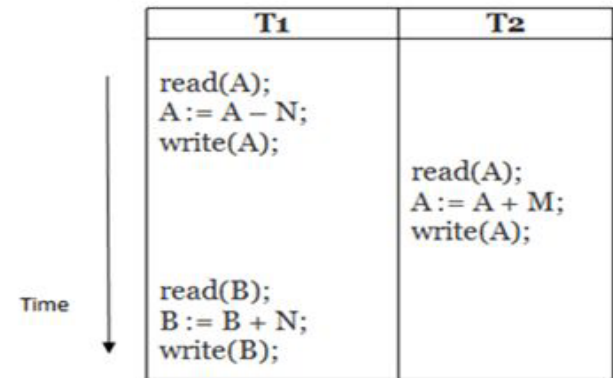
- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.

(c)



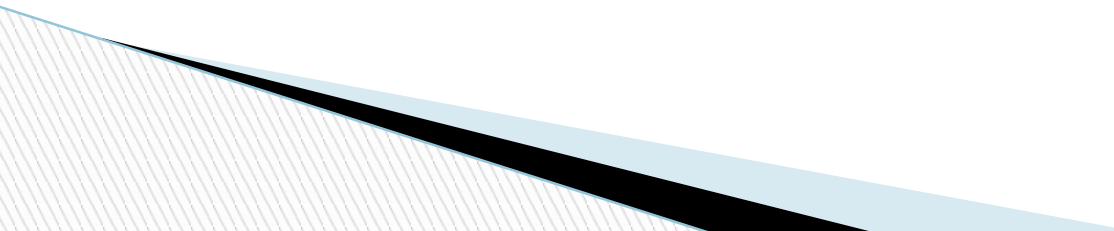
Schedule C

(d)



Schedule D

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
 - It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
 - A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.
- 

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

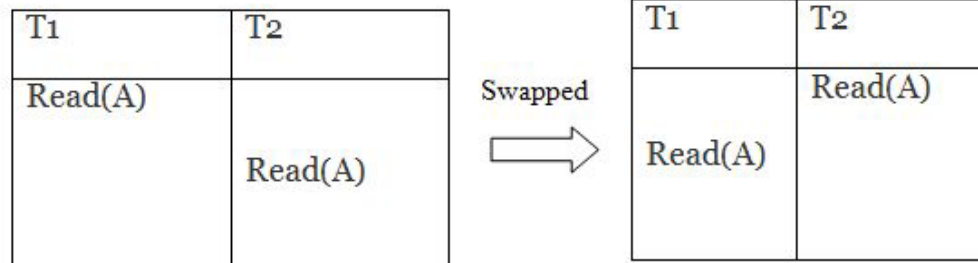
Conflicting Operations

The two operations become conflicting if all conditions satisfy:

- Both belong to separate transactions.
- They have the same data item.
- They contain at least one write operation.

Example: Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)

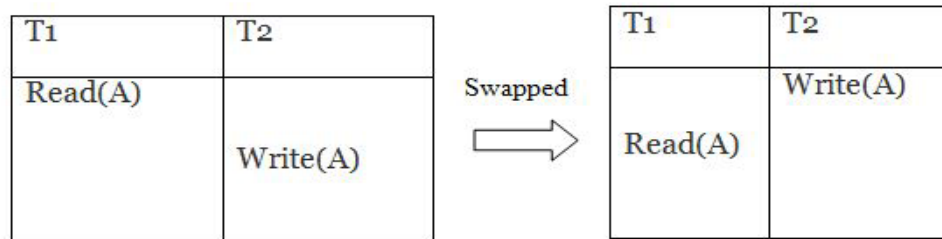


Schedule S1

Schedule S2

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



Schedule S1

Schedule S2

Here, S1 ≠ S2. That means it is conflict.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	Write(A)

T1	T2
Read(A)	Write(A)

Two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

Schedule S1

Schedule S2

2. Updated Read

In schedule S1, if T_i is reading A which is updated by T_j then in S2 also, T_i should read A which is updated by T_j .

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	<u>Read(A)</u>

Schedule S2

Two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

□ **Example: Schedule S**

T1	T2	T3
Read(A)	Write(A)	Write(A)
Write(A)		

With 3 transactions, the total number of possible schedule

$$= 3! = 6$$

S1 = <T1 T2 T3>

S2 = <T1 T3 T2>

S3 = <T2 T3 T1>

S4 = <T2 T1 T3>

S5 = <T3 T1 T2>

S6 = <T3 T2 T1>

Taking first schedule S1:

□

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Hence, view equivalent serial schedule is:

T1 → T2 → T3

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Cascaded Aborts

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Rollback or Cascading Abort or Cascading Schedule. It simply leads to the wastage of CPU time. These Cascading Rollbacks occur because of **Dirty Read problems**.

T1	T2	T3	T4
R(A)			
W(A)			
	R(A)		
	W(A)		
		R(A)	
		W(A)	
			R(A)
			W(A)
Failure			

Cascading Recoverable Schedule

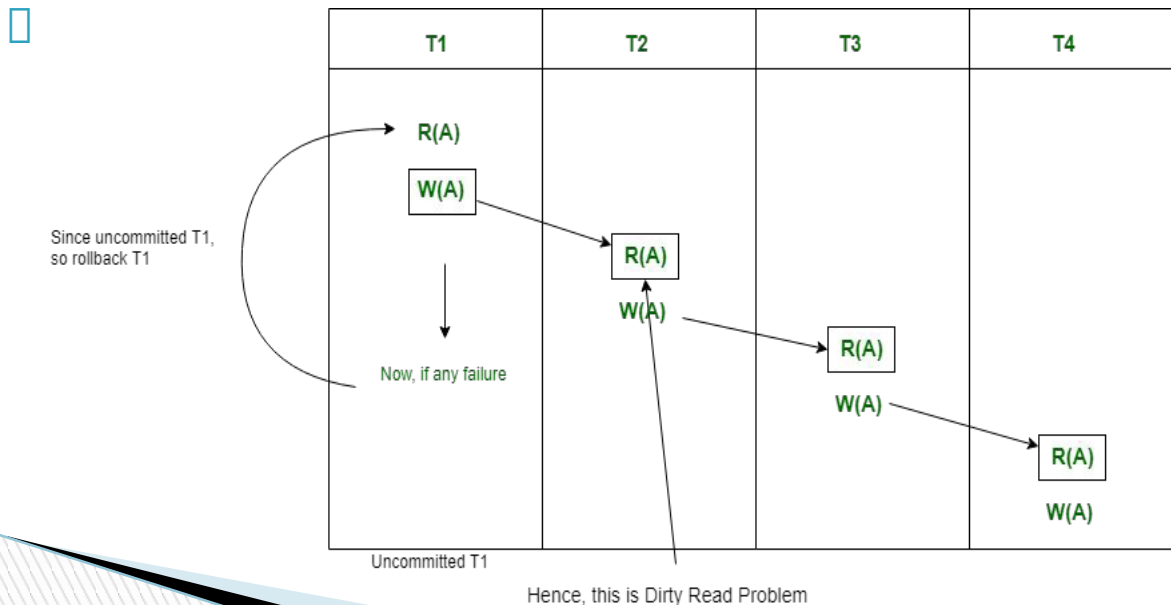
T1	T2	T3
R(A)		
W(A)		
Commit		
	R(A)	
	W(A)	
	Commit	
		R(A)
		W(A)
		Commit

Cascadeless Schedule

- For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3.

Suppose at this point T1 fails.

T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.



Because of T1 rollback, all T2, T3, and T4 should also be rollback (Cascading dirty read problem).

This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called **Cascading rollback**.

Recoverable and Nonrecoverable Schedules

- **Recoverable schedule:** One where no *committed* transaction needs to be rolled back (aborted).

A schedule *S* is **recoverable** if no transaction *T* in *S* commits until all transactions *T'* that have written an item that *T* reads have committed.

- **Non-recoverable schedule:** A schedule where a committed transaction may have to be rolled back during recovery.

This violates **Durability** from ACID properties (a committed transaction cannot be rolled back) and so non-recoverable schedules *should not be allowed*.

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

Table shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if T_j reads the updated value of T_i and T_j committed before T_i commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if T_j reads the updated value of T_i. Commit of T_j is delayed till commit of T_i.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

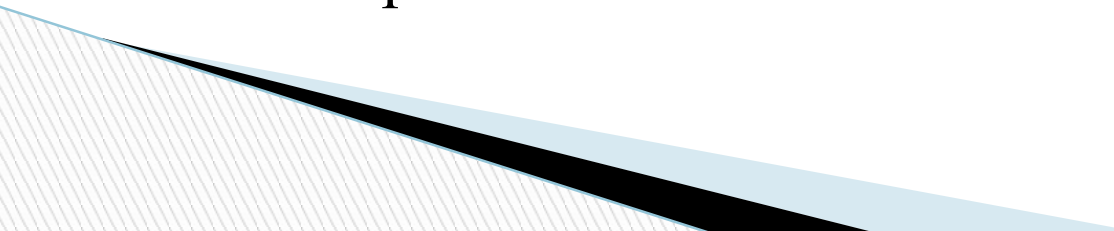
The above Table shows a schedule with two transactions. Transaction T₁ reads and write A and commits, and that value is read and written by T₂. So this is a cascade less recoverable schedule.

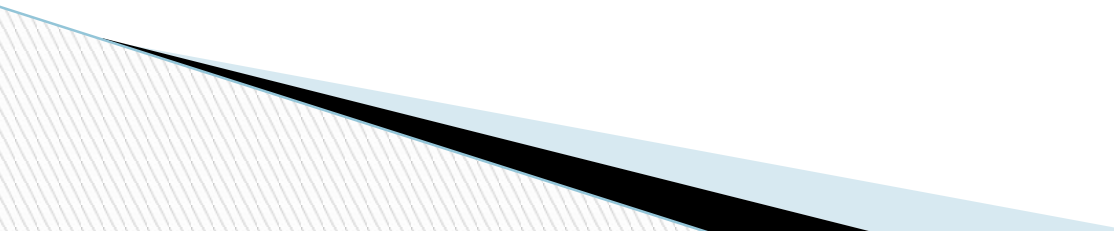
Concurrency Control

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

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.



- ❑ **Locking Methods** in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.
 - ❑ A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.
 - ❑ All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.
- 

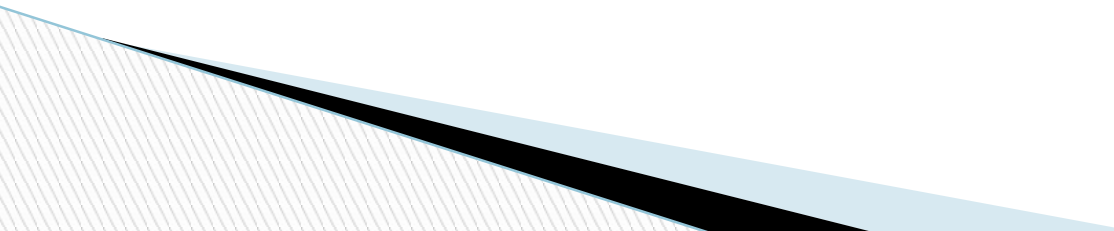
Lock Based Protocol

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 (S):

A shared lock is also.

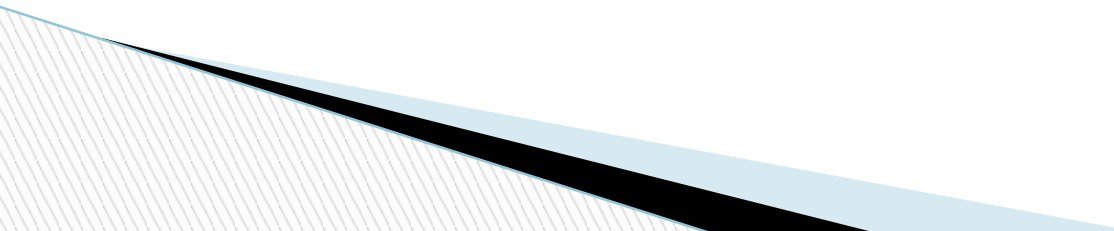
For example, concalled a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data items. Consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.



2. Exclusive Lock (X):

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. You can allow this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevents this operation.



□ 3.Simplistic Lock Protocol

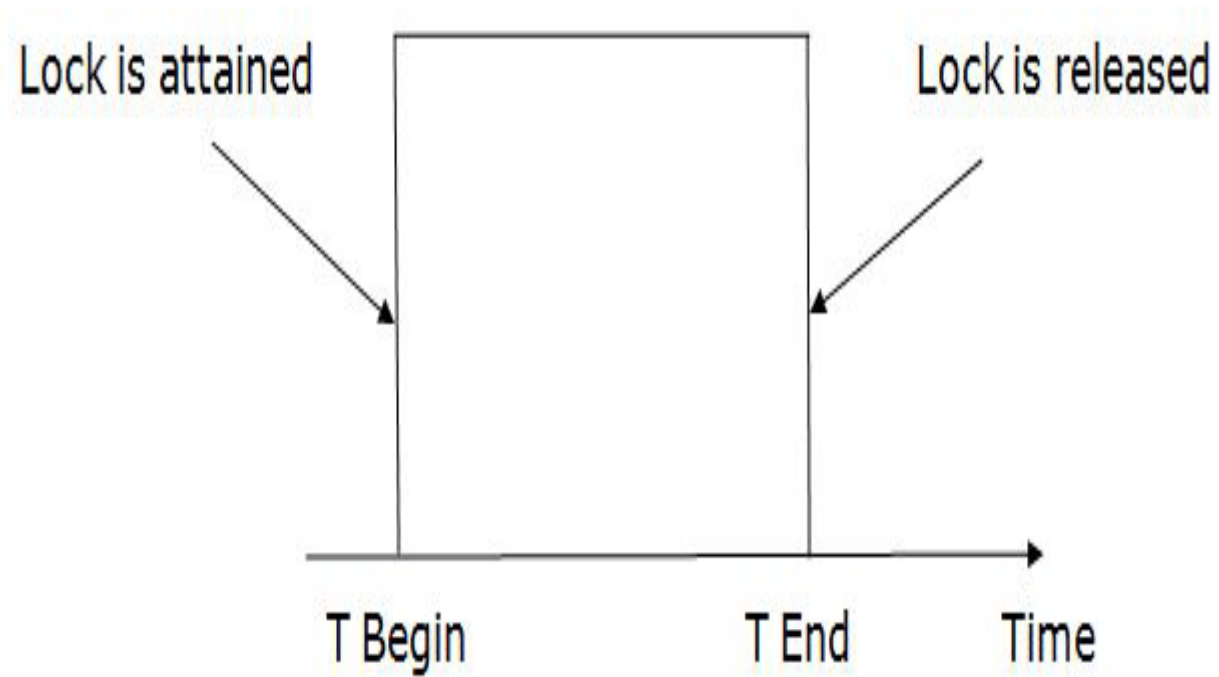
- It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

❑ 4. Pre-claiming Locking 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.

❑



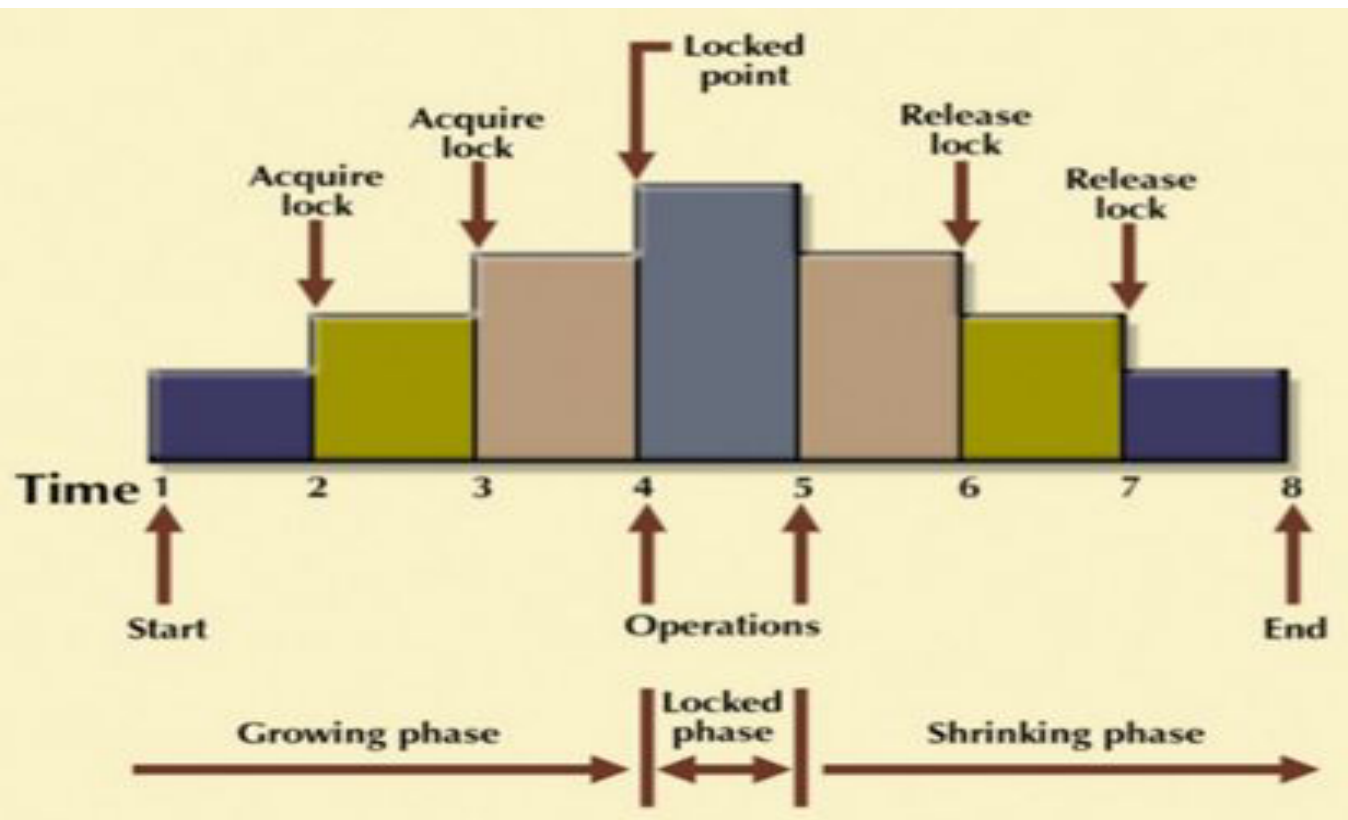


Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.

This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

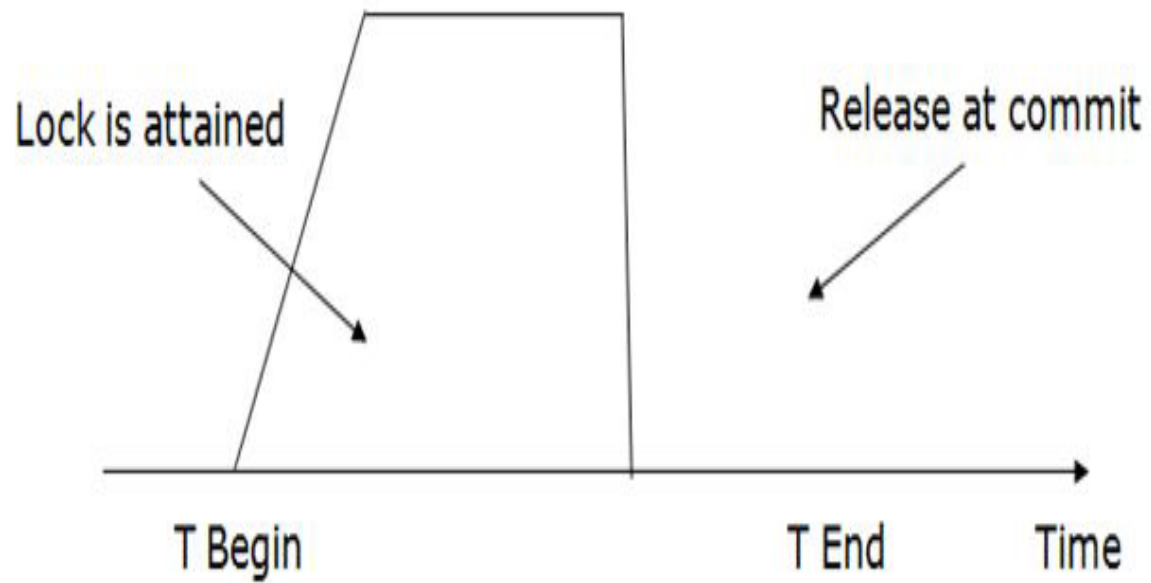
Growing Phase: In this phase transaction may obtain locks but may not release any locks.

Shrinking Phase: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

❑ 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.
- ❑

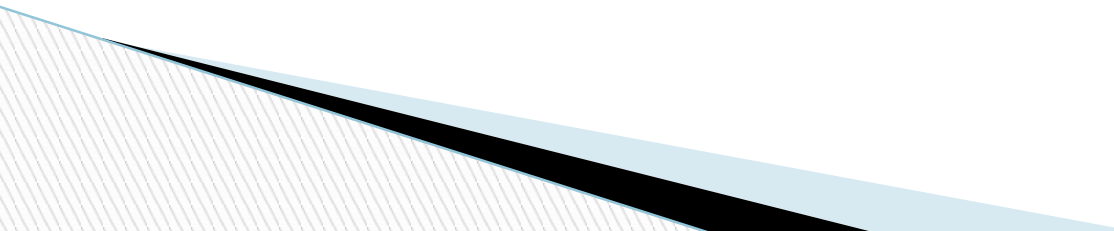


It does not have cascading abort as 2PL does.

Starvation

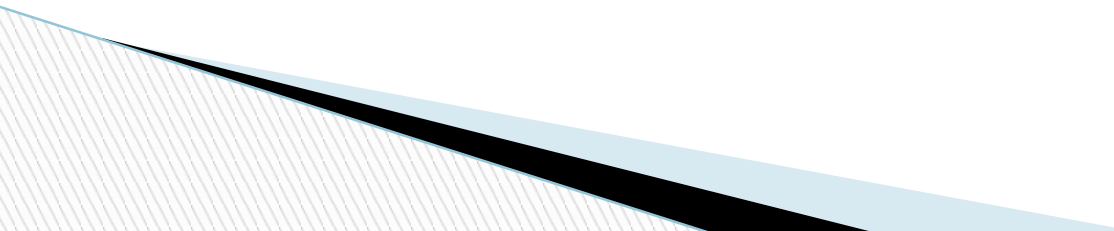
Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Following are the reasons for Starvation:

- ❑ When waiting scheme for locked items is not properly managed
 - ❑ In the case of resource leak
 - ❑ The same transaction is selected as a victim repeatedly
- 

Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.
- 

- **For example:** In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.
- Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

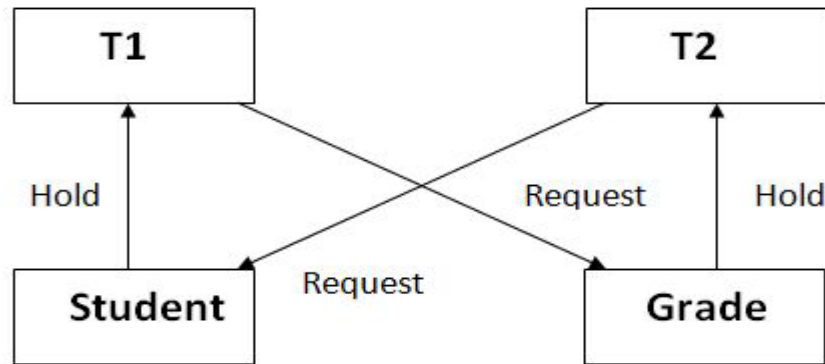


Figure: Deadlock in DBMS

Deadlock Avoidance

When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.

Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.



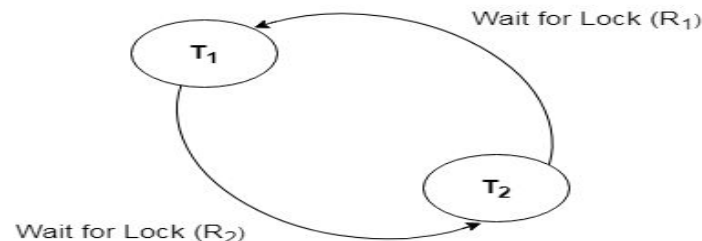
Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

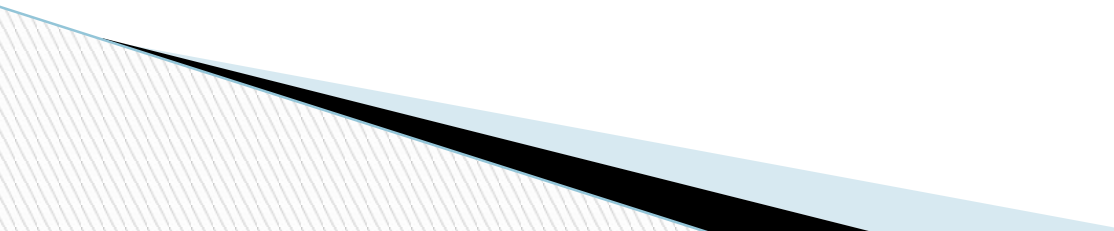
Wait for Graph

This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.

The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
 - The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.
- 

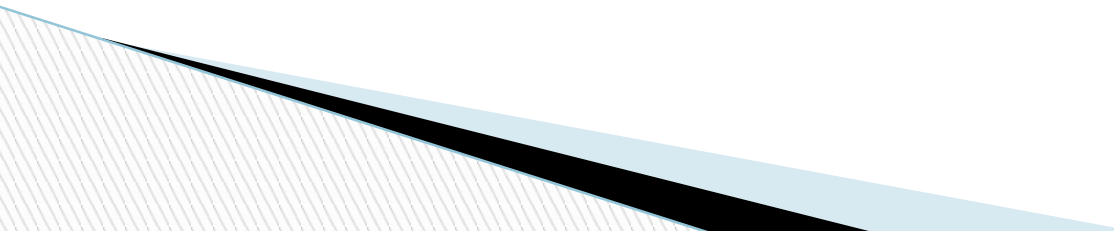
Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions T_i and T_j and let $TS(T)$ is a timestamp of any transaction T . If T_2 holds a lock by some other transaction and T_1 is requesting for resources held by T_2 then the following actions are performed by DBMS:

- Check if $TS(T_i) < TS(T_j)$ - If T_i is the older transaction and T_j has held some resource, then T_i is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
- Check if $TS(T_i) > TS(T_j)$ - If T_i is older transaction and has held some resource and if T_j is waiting for it, then T_j is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
 - If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.
- 

Recovery methods

- ▣ **Shadow-Paging**
 - ▣ **Log-Based Recovery**
 - ▣ **Checkpoints**
- 

Shadow-Paging

Now let see the concept of shadow paging step by step –

Step 1 – Page is a segment of memory. Page table is an index of pages. Each table entry points to a page on the disk.

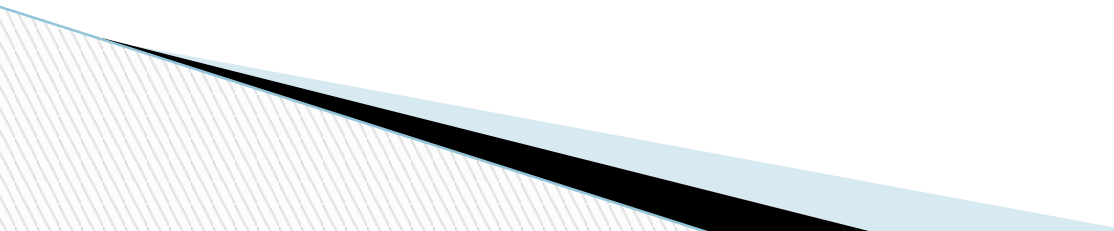
Step 2 – Two page tables are used during the life of a transaction: the current page table and the shadow page table. Shadow page table is a copy of the current page table.

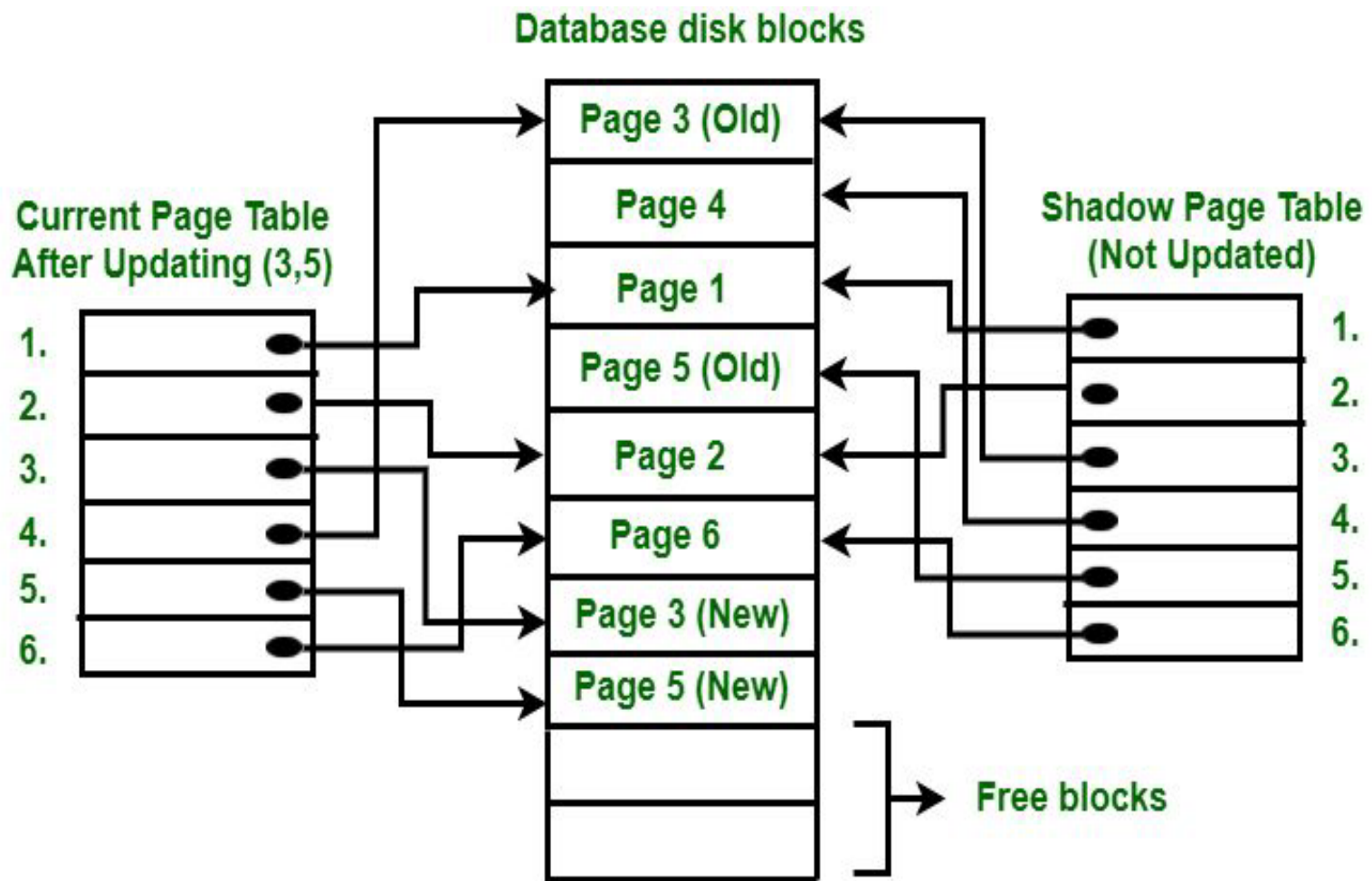
Step 3 – When a transaction starts, both the tables look identical, the current table is updated for each write operation.

Step 4 – The shadow page is never changed during the life of the transaction.

Step 5 – When the current transaction is committed, the shadow page entry becomes a copy of the current page table entry and the disk block with the old data is released.

Step 6 – The shadow page table is stored in non-volatile memory. If the system crash occurs, then the shadow page table is copied to the current page table.

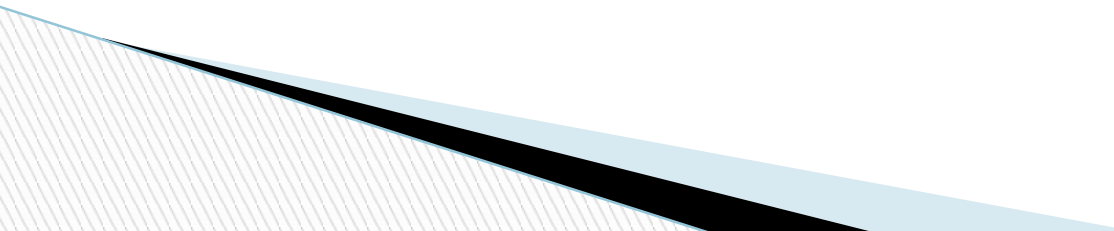




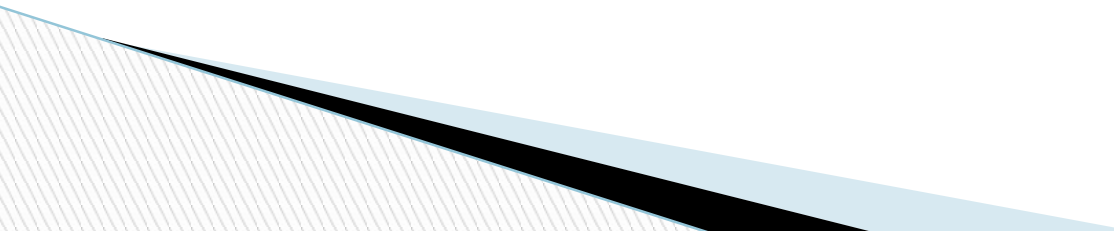
Advantages

- ❑ The advantages of shadow paging are as follows –
- ❑ No need for log records.
- ❑ No undo/ Redo algorithm.
- ❑ Recovery is faster.

Disadvantages

- ❑ The disadvantages of shadow paging are as follows –
 - ❑ Data is fragmented or scattered.
 - ❑ Garbage collection problem. Database pages containing old versions of modified data need to be garbage collected after every transaction.
 - ❑ Concurrent transactions are difficult to execute.
- 

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
 - If any operation is performed on the database, then it will be recorded in the log.
 - But the process of storing the logs should be done before the actual transaction is applied in the database.
- 

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

When the transaction is initiated, then it writes 'start' log.

<Tn, Start>

When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

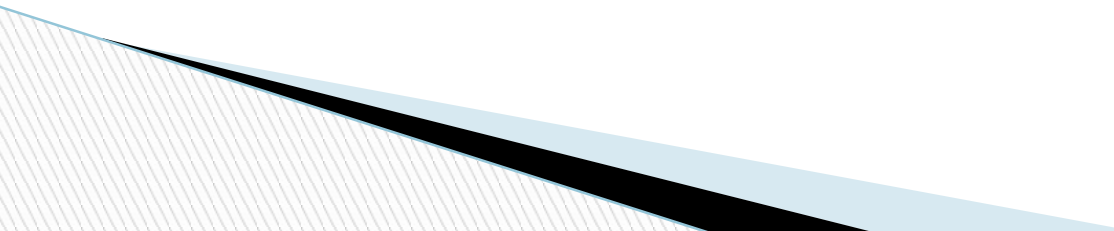
<Tn, City, 'Noida', 'Bangalore' >

When the transaction is finished, then it writes another log to indicate the end of the transaction.

<Tn, Commit>



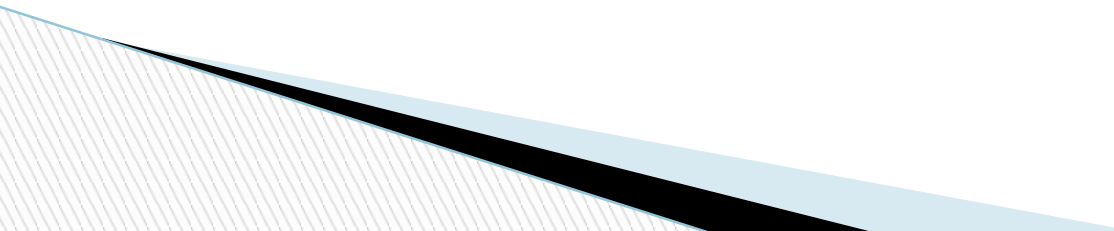
Recovery using Log records

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
 - If any operation is performed on the database, then it will be recorded in the log.
 - But the process of storing the logs should be done before the actual transaction is applied in the database.
- 

- Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.
- When the transaction is initiated, then it writes 'start' log.
- $\langle T_n, \text{Start} \rangle$
- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.
- $\langle T_n, \text{City}, 'Noida', 'Bangalore' \rangle$
- When the transaction is finished, then it writes another log to indicate the end of the transaction.
- $\langle T_n, \text{Commit} \rangle$

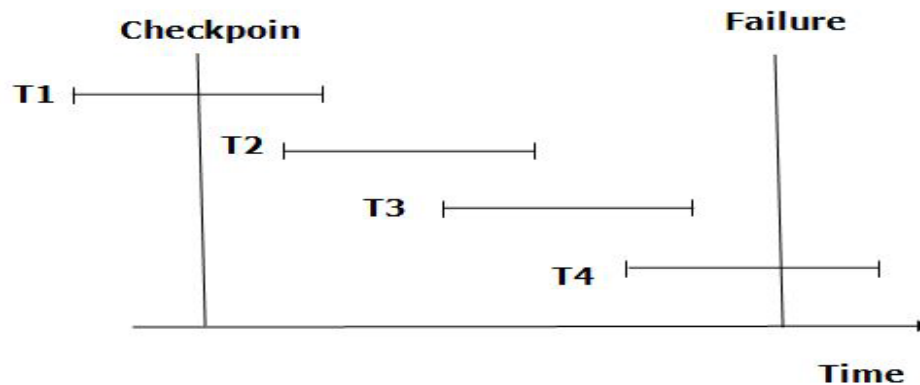
- When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.
- If the log contains the record $\langle T_i, \text{Start} \rangle$ and $\langle T_i, \text{Commit} \rangle$ or $\langle T_i, \text{Commit} \rangle$, then the Transaction T_i needs to be redone.
- If log contains record $\langle T_n, \text{Start} \rangle$ but does not contain the record either $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$, then the Transaction T_i needs to be undone.

Checkpoints

- A checkpoint is a process that saves the current state of the database to disk. This includes all transactions that have been committed, as well as any changes that have been made to the database but not yet committed.
 - The checkpoint process also includes a log of all transactions that have occurred since the last checkpoint. This log is used to recover the database in the event of a system failure or crash.
 - When a checkpoint occurs, the DBMS will write a copy of the current state of the database to disk. This is done to ensure that the database can be recovered quickly in the event of a failure.
- 

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.

- ❑ **For example:** In the log file, transaction T2 and T3 will have $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$. The T1 transaction will have only $\langle T_n, \text{commit} \rangle$ in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- ❑ The transaction is put into undo state if the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- ❑ **For example:** Transaction T4 will have $\langle T_n, \text{Start} \rangle$. So T4 will be put into undo list since this transaction is not yet complete and failed amid.