Unit-5

Transaction → It is a set of logically related operations. For ex→ you are transferring money from your bank account to your friend's account, the set of operations would be like this –

Simple Transaction

1) Read your account balance
2) Deduct the amount from your balance
3) Write the remaining balance to your account.
4) Read your friend's account balance
5) Add the amount to his account balance
6) Write the new updated balance to his account.

This whole set of operations can be called a transaction.

or

→ It is a set of operations used to perform a logical unit of work.

→ transaction generally represent change in database.

Two operations of transaction —

① Read ② Write

↓ ↓

Database access change

Read(A) ⇒ accessing the dataitem from DB to main memory.

Write(A) ⇒ updation of dataitem to DB.

Transfer

R(A)

A = A-500

W(A)

R(B)

B = B+500

W(B)

Commit

A = ~~1000~~ 500

B = ~~2000~~ 2500

// after commit change in database happen otherwise all the operations are performing in RAM only.

ACID properties of transaction

1) Atomicity ② Consistency ③ Isolation

④ Durability

Atomicity → Atomicity states that either all the transaction are executed or none are executed.

(2) Consistency → After the execution of transaction the database must be in consistent formed.

* (Before transaction start and after the transaction completed sum of money should be same)

T_1	
R(A)	⇒ 2000
A = A − 1000	⇒ 1000
W(A)	⇒ 1000
R(B)	⇒ 3000
B = B + 1000	⇒ 4000
W(B)	⇒ 4000
commit	

5000 [1000, 4000]

A = 2000
B = 3000

(3) Isolation →
(when parallel transactions are executing then they are intefering because CPU is switching over)
(It says can I convert parallel schedule in serial schedule)

If 2 processes enter in transaction for execution, then both process don't disturb each other.
for eg → T_i & T_j are two transactions
either T_i finished before T_j
or
T_j finished before T_i

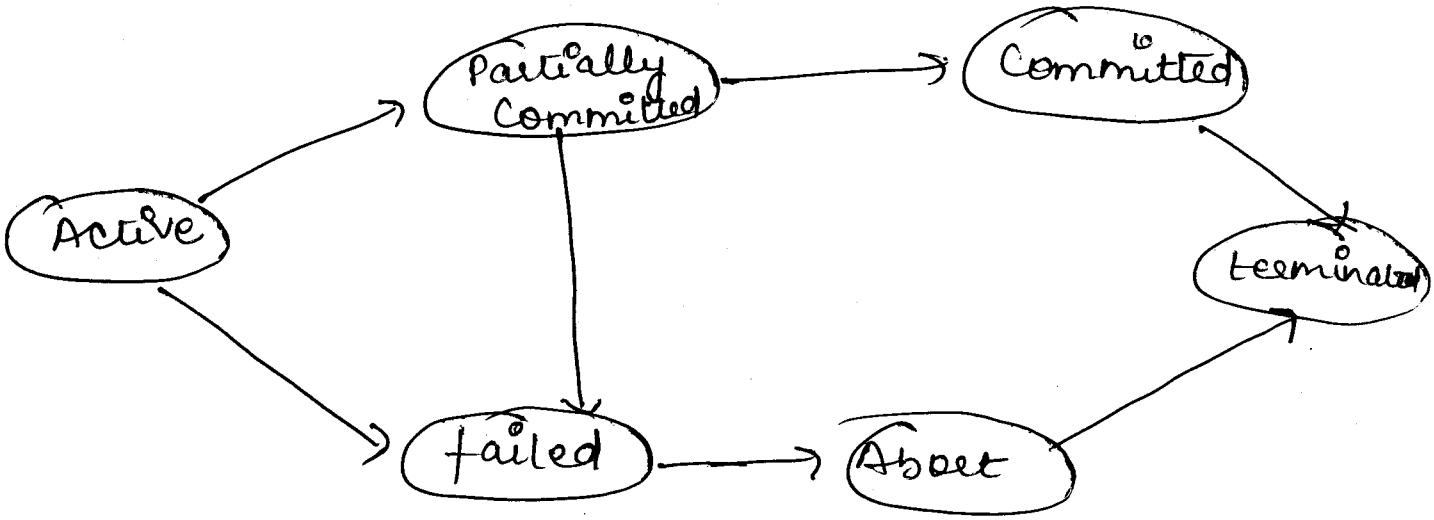
$T_1 \rightleftharpoons T_2$
!! convert it
$T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$

A = 5000

T_1	T_2
R(A)	
A = A − 1000	
	R(A)
	A = A + 500
W(A)	
	W(A)
4000	5500

(4) Durability ⇒ After transaction completed successfully, the change it has made to database permanent if there are system failure.

Transaction State



Active → The initial state means the transaction is in execution.

Partially Committed → After the execution of the last operation of transaction. The transaction will be in partially committed state.

Committed state → when all modification done after the execution of the final instruction.

Failed → after discovery that normal execution can no longer proceed; (due to some logical error failed)

Aborted → After transaction has been rolled back and the database restored to it previous state to start transaction. 2 options after aborted —

→ restart transaction, can be done only if no terminal logical error

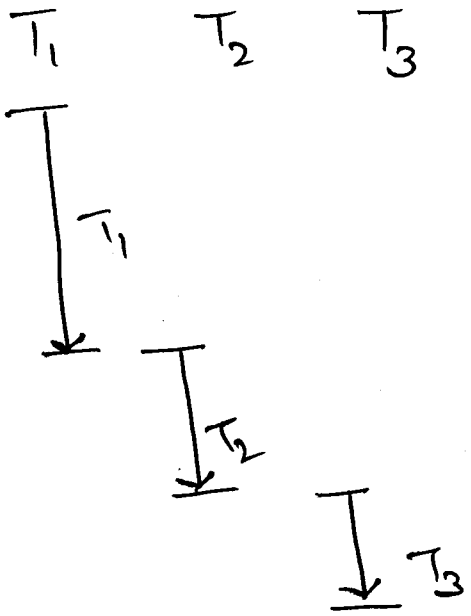
→ kill the transaction.

Types of schedule → Serial
→ Concurrent

Schedule → It is chronological execution sequence of multiple transactions.

T_1 T_2 T_3 - - - T_n

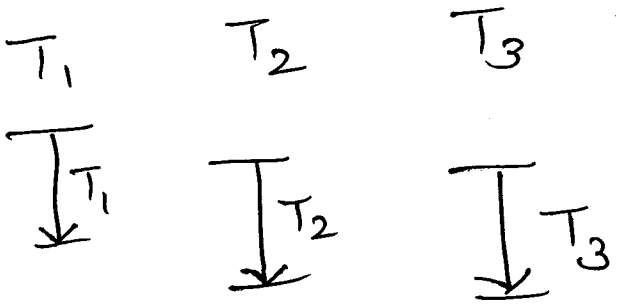
Serial



(2) After commit of one transaction, only then start the other transaction.

(3) Advantage of serial schedule is consistency. Database is always consistent.

Parallel



(2) Interleaved execution or simultaneous execution of 2 or more transaction.

ex →

T_1	T_2
$R_1(A)$	
$W_1(A)$	$R_2(A)$
	$R_2(B)$
$R_1(B)$	
$W_1(B)$	

Inconsistent schedule

ex→

T_1	T_2
R1(A)	
W1(A)	
R1(B)	
W1(B)	
	$R_2(A)$
	$R_2(B)$

(3) Disadvantage of serial transaction is throughput of system is very less.
(first transaction complete after that 2nd transaction will execute)

To remove this disadv. we use parallel schedule

(4) Performance is low

(3) Advantage of parallel or concurrent schedule
↳ Throughput (increases)

(4) Disadvantage is inconsistent schedule. (concurrent execution of two or more than two transaction use concurrency control techniques.

SCHEDULE

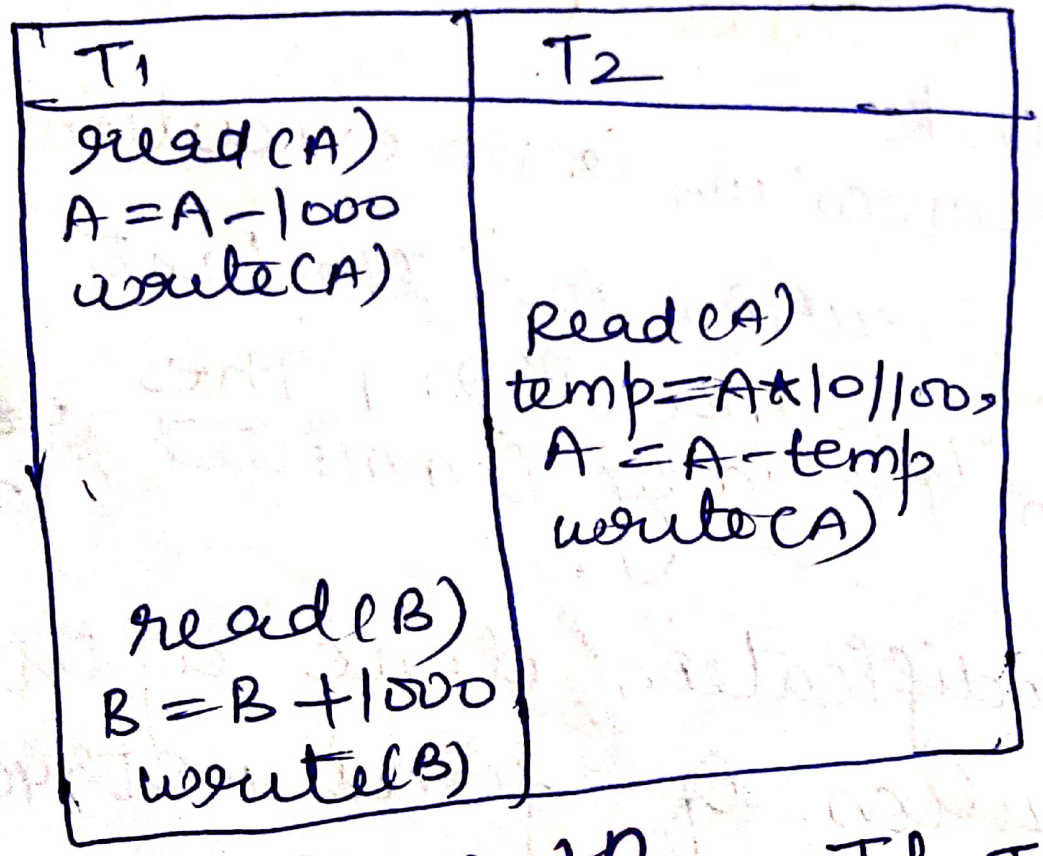
The Schedule Contains the set of the transcation

1) Serial
2) Concurrent

Serial Schedule :- In serial schedule all the statements of the transcation T_i must be finished. Thereafter the operation of T_j will be allowed to execute.

T_1	T_2
read (A)	
A = A − 1000	
write (A)	
read (B)	
B = B + 1000	
write (B)	
	read (A)
	temp = A * 10/100
	A = A − temp;
	write (A)

Concurrent schedules -



Conflicit Instns — If $I_i \in T_i$, $I_j \in T_j$

1) I_i = read(Q) , I_j = write(Q)
2) I_i = write(Q) , I_j = read(Q)
3) I_i = write(Q) , I_j = write(Q)

Let us Consider Ii & Ij are 2 Consecutive Instn of transaction Ti, Tj Respectively. Assume Both the instruction work on the same data item Q, then these instn are said to be conflicting Instn if ~~both~~ one of following Condn will be true:

Ii: read(Q) Ij = write(Q)
Ii = write(Q) Ij = read(Q)
Ii = write(Q) Ij = write(Q)

For Conflicting Instn, atleast one of the operation is write(Q) operation.

Conflict Equivalent Schedule :—

Two Schedules S & S' are called Conflict Equivalent if S' is obtained by swapping a series of non-Conflicting Instn.

S

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

S'

T_1	T_2
r(A)	
w(A)	
	r(A)
r(B)	
	w(A)
w(B)	
	r(B)
	w(B)

S''

T_1	T_2
r(A)	
w(A)	
	r(A)
~~r(B)~~	
w(B)	
	w(A)
	r(B)
	w(B)

S'''

T_1	T_2
r(A)	
w(A)	
r(B)	
	r(A)
w(B)	
	w(A)
	r(B)
	w(B)

S''''

T_1	T_2
r(A)	
w(A)	
r(B)	
w(B)	
	r(A)
	w(A)
	r(B)
	w(B)

S, S', S'', S''', S''''
all are Conflict
Equivalent Schedu

Conflicit serilizability Schedule:-

A Schedules is called conflicit seriliziable Schedule if S is conflicit Equivalent to a some serial schedule.

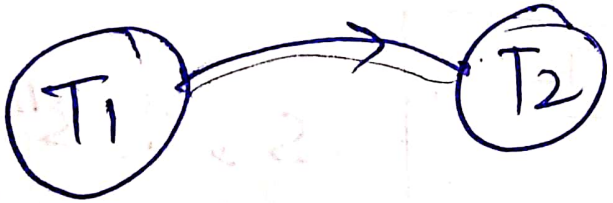
Test for Conflicit serilziability :-

1) Prepare a preceedence graph for the gives schedul
2) Determine a cycle exist or not within the graph.
   a) if a cycle is detected it indicate the gives schedule is not conflicit serilizable schedule.
   b) otherwise conflicit serilizable.

Edge in precedence graph:-

An edge $T_i \longrightarrow T_j$ will exist within the precedence graph if atleast one of 3 cond" hold.

i) T_i executes read Q before T_j executes write Q.
2) " " write(Q) " " " read(Q)
3) " " write(Q) " " " write(Q)

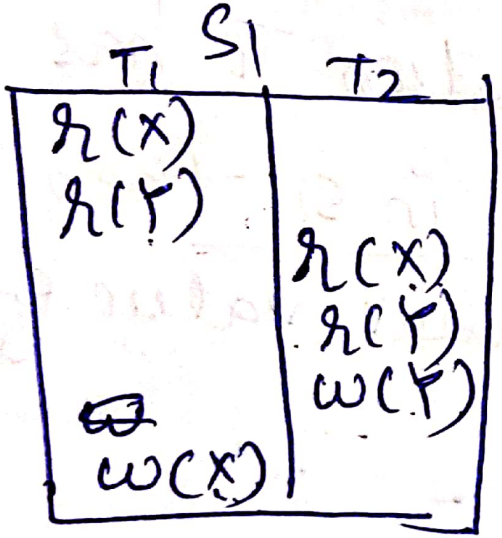


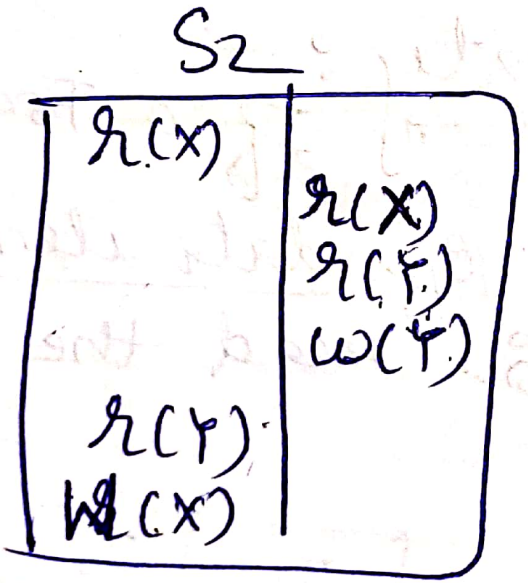
no cycle
conflict serializable.

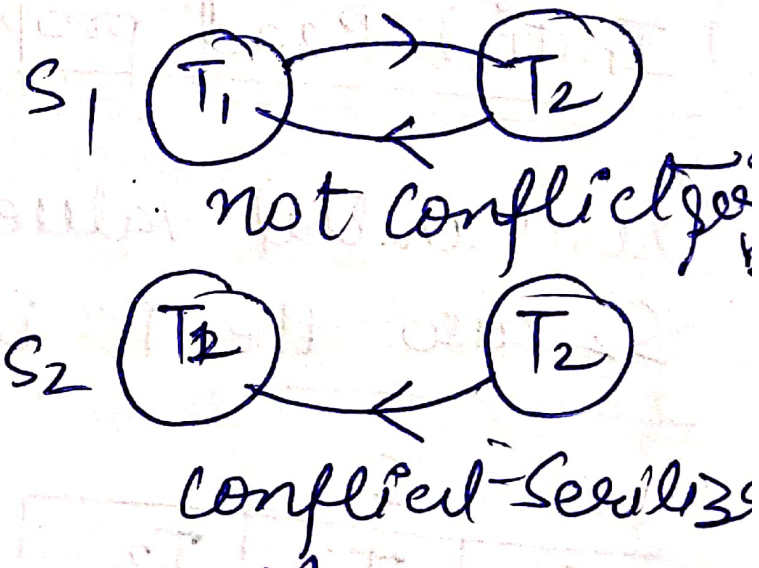
Q: Consider the following schedule involving Trans"

S1: $r_1(X), r_1(Y), r_2(X), r_2(Y), w_2(Y), w_1(X)$

S2: $r_1(X), r_2(X), r_2(Y), w_2(Y), r_1(Y), w_1(X)$







a) Both S1 & S2 are conflict serlizable.

b) S1 is conflict serlizable but S2 not

c) S1 is not " but S2 is conflictserilizal

d) Both are not

Q Consider two transaction T_1 & T_2 with 4 schedule given below:-

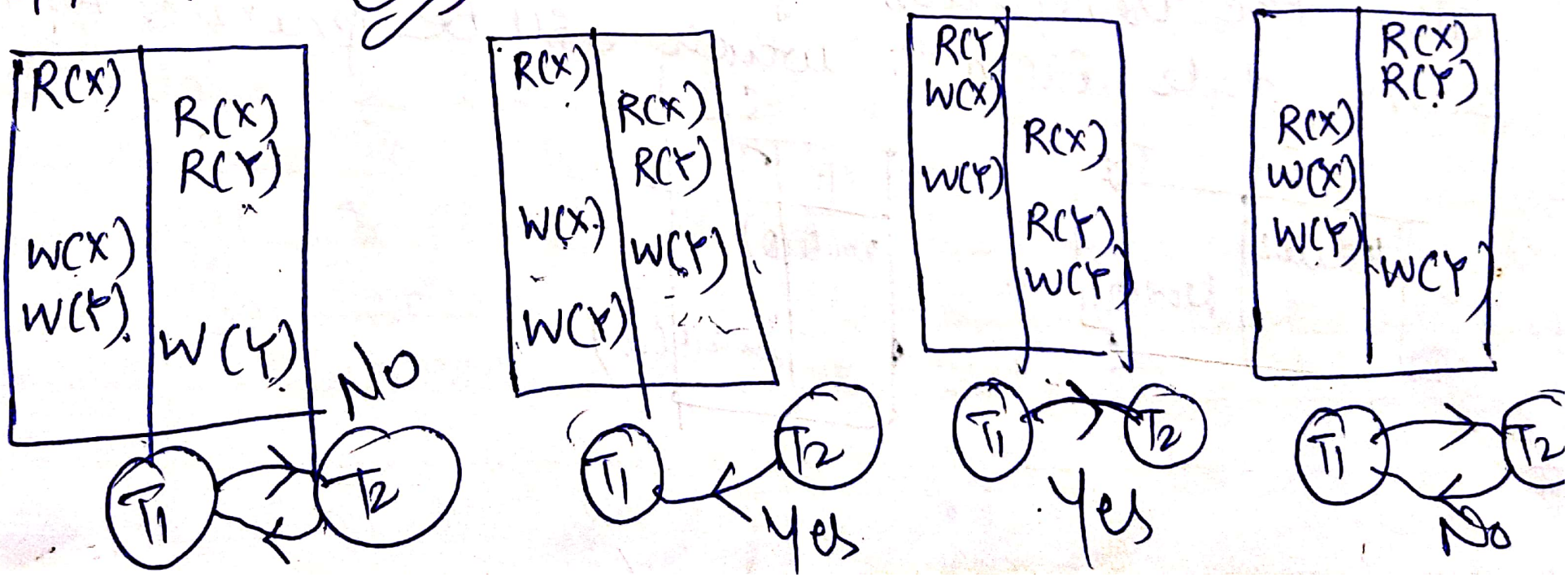
S1: $R_1(X)$, $R_2(X)$, $R_2(Y)$, $W_1(X)$, $W_1(Y)$, $W_2(Y)$

$S_2 = R_1(X)$, $R_2(X)$, $R_2(Y)$, $W_1(X)$, $W_2(Y)$, $W_1(Y)$

$S_3 = R_1(Y)$, $W_1(X)$, $R_2(X)$, $W_1(Y)$, $R_2(Y)$, $W_2(Y)$

$S_4 = R_2(X)$, $R_2(Y)$, $R_1(X)$, $W_1(X)$, $W_1(Y)$, $W_2(Y)$

a) S_1, S_2 ~~b)~~ S_2, S_3 c) S_3 d) S_4



View Equivalent Schedule :-

Two schedules S_1 & S_2 are called view equivalent if they satisfy following 3 cond^n!

i) **Initial Read property :-** If a Transcation T_i has read initial value of data item Q in S_1 then S_2 also the T_i will read the initial value of Q.

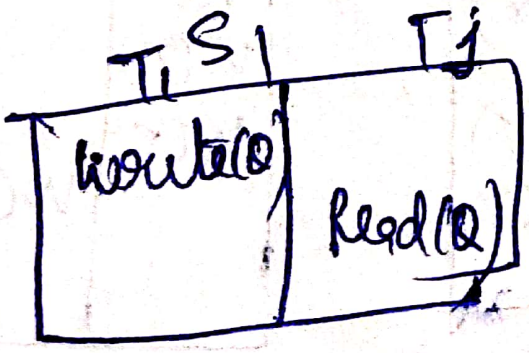
S_1

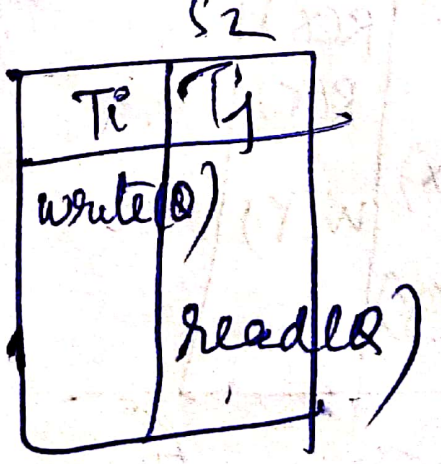
T_1	T_2	T_3
1 $r(A)$		
	$w(A)$	
$w(A)$		
		2 $w(A)$

T_1	T_2	T_3
1 $r(A)$		
$w(A)$		
	$w(A)$	
		2 $w(A)$

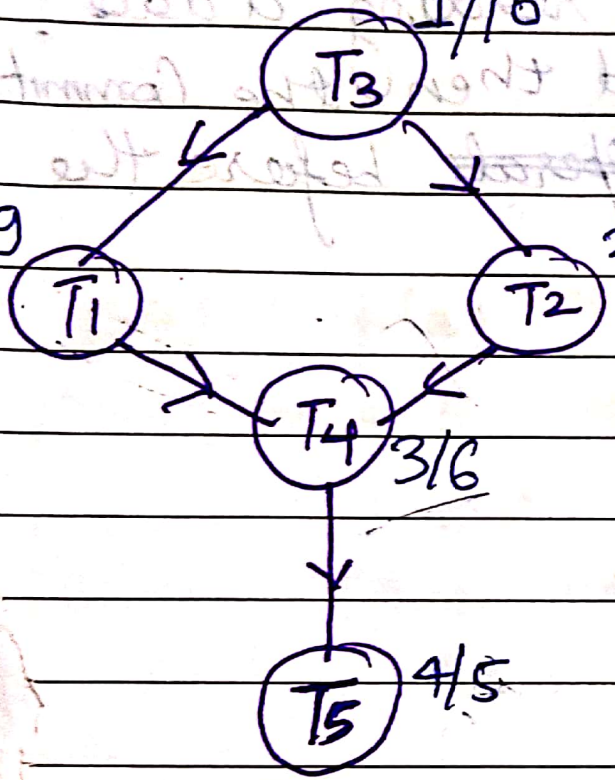
2) Final Write property :— If a transcation T_i perform the final write operation on Q in S_1 then T_i will perform the final write operation of Q in S_2, as well.

3) W.R Conflict property :— In S_1 if the transcation T_j reads the data item Q. ie produce By Transcation T_i. Then in S_2 also the transcation T_j will also read the same data item Q which will be produce by





How to find the sequence of Execution of Transcation if the precedence graph is given :— Topological Sorting



Order of Execution :—
write the Max finish time first.

$T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_5$

OR $T_3 \rightarrow T_2 \rightarrow T_1 \rightarrow T_4 \rightarrow T_5$

If the given schedule is Conflict serializable ie not contain any cycle. Then can only we find the order of Execution of Transcation. Apply Topological sorting.

can apply.

Topological sort :- i) direct graph
ii) Acyclic graph

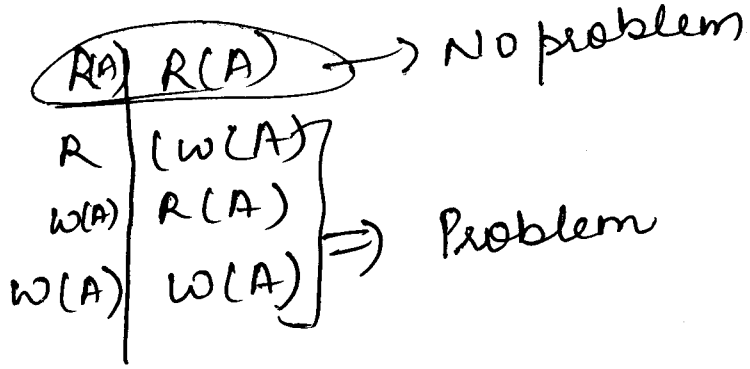
View Serializable Schedule :-

A Schedule S is called view serializable if it is view equivalent to a serial schedule.

Key point :-

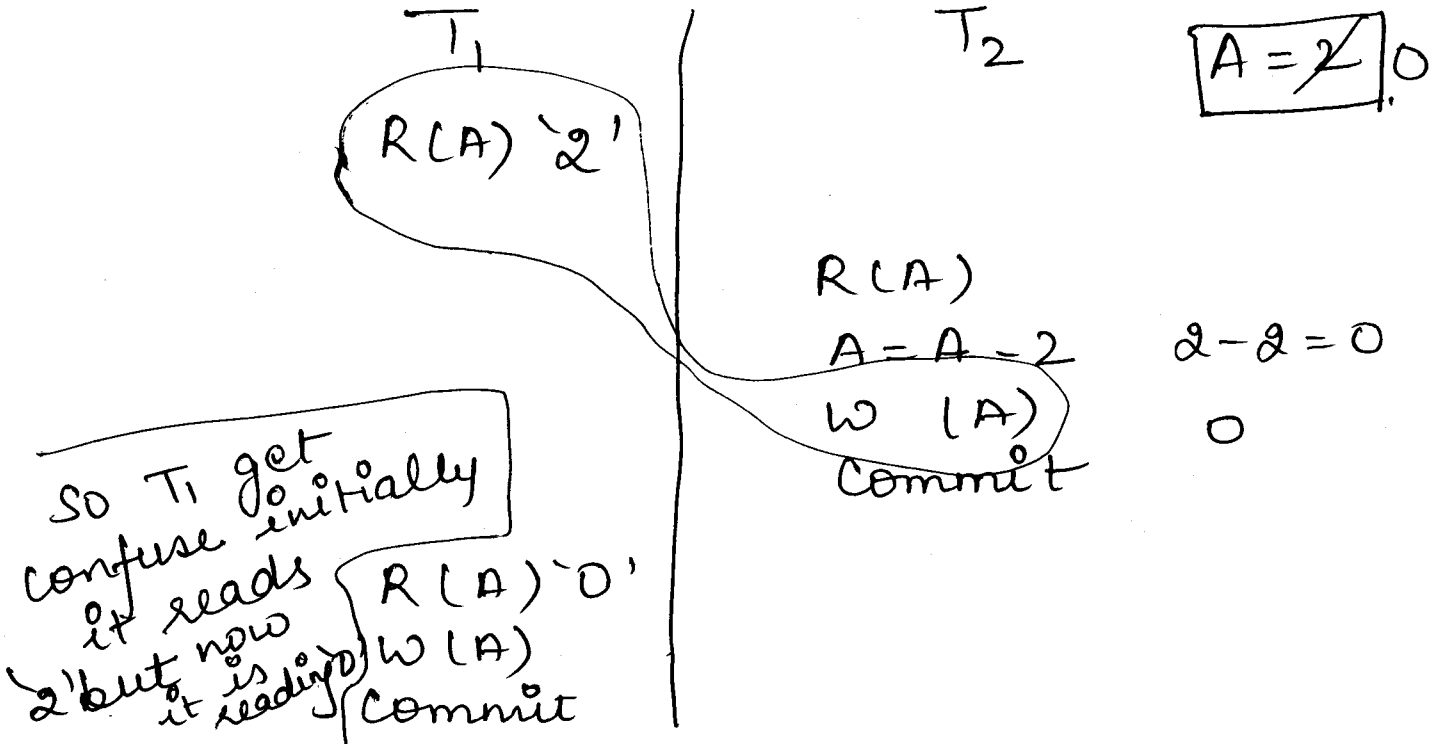
1) Checking the view serializability of a schedule is a NP hard problem.
2) All Conflict serializable schedule are also view serializable schedules but vice versa is not true.

Problem with concurrent schedule

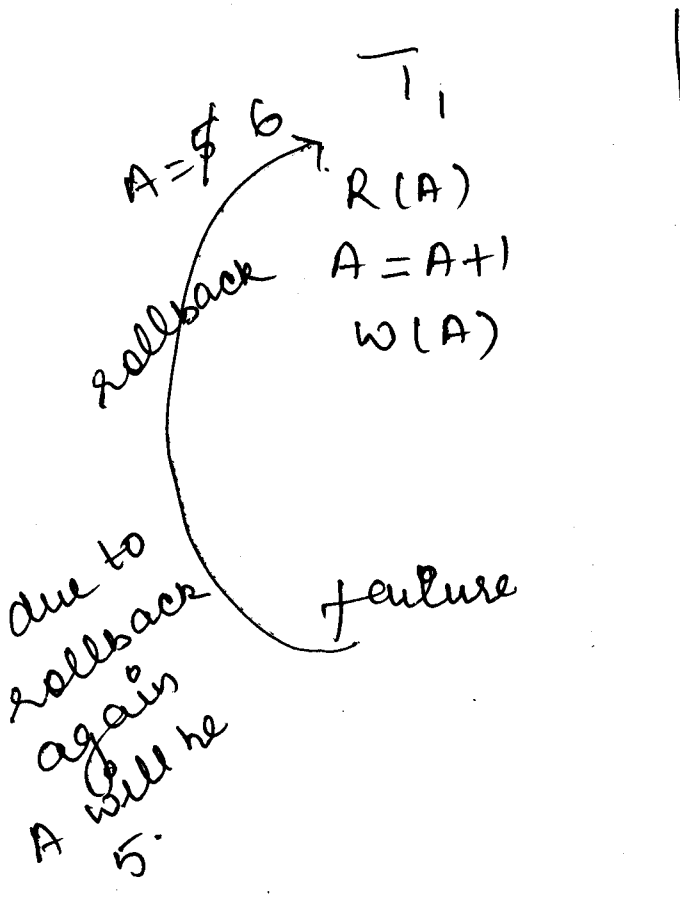


① Read Write Conflict or Unrepeatable read

If a transaction 'Ti' reads an item value twice and the item is changed by another transaction 'Tj' in between the 2nd read operation. Hence 'Ti' receives different values for its 2nd read operation of the same item and can lead to confusion.



② Temporary update (dirty read) problem or write-read conflict



(because transaction T_2 read the value which is uncommited) This dirty read problem.

(there is a problem if you read value which is not committed after that due to some problem transaction rollback then you don't have a chance to rollback.)

(If the running transaction read the value of uncommited transaction then it is know as dirty read problem).

(3) Lost Update Problem or Write-Write conflict problem or Blind write

A = 40

T_1 : R(A)

T_2 : A = A + 10

T_1 : W(A) → 50

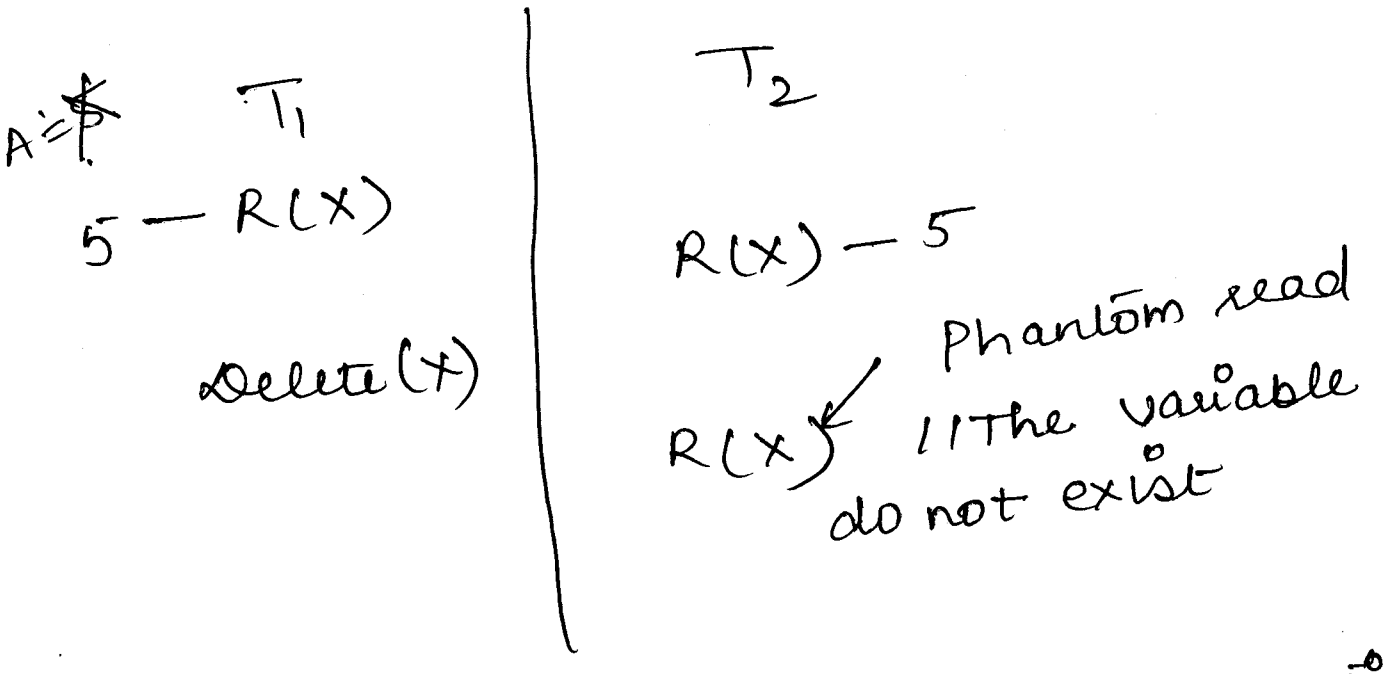
T_2 : W(A) → 50 (Blind write)

T_2 : Commit

But T_1 update commit so it is lost.

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Phantom Read problem



This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable doesn't exist.

→ Recovery from Transaction failures:—

⑥ A transacⁿ has to abort when it fails to execute or when it reaches a point where it can't go any further. This is called transacⁿ failure where only a few transac's or processes are hurt.

⑥ Reasons for transacⁿ failure could be:—

→ Logical errors: Where a transacⁿ cannot complete bcoz it has some code error or any internal error condition.

→ System errors: where the database system itself terminates an active transacⁿ bcoz ~~of some system~~ the DBMS is not able to execute it, or it has to stop bcoz of some system condⁿ, for eg: in case of deadlock or resource unavailability, the system aborts an active transacⁿ.

(it is optional)

storage space:

→ Volatile: can't survive sys. crashes, eg M.M, cache mem. They are fast but can store only a small amt. of info.

→ Non-volatile: can survive sys. crashes eg: hard-disks, magnetic tapes, flash mem & non-volatile RAM. They are huge in data storage capacity, but slower in accessibility.

⑥ When a DBMS recovers from a crash, it should maintain the following

(1) It should check the states of all the transac's, which were being executed.

(2) A transacⁿ may be in the middle of some operaⁿ; the DBMS must ensure the atomicity of the transacⁿ in this case.

(3) It should check whether the transacⁿ can be completed now or it needs to be rolled back.

(4) No transacⁿs would be allowed to leave the DBMS in an inconsistent state.

(5) These are two types of techniques to recover from transacⁿ failures —

- (1.) Log based Recovery
- (2.) checkpoints
- ~~(3.) Backup mechanism.~~
- (4.) Shadow Paging

→ LOG-BASED RECOVERY :-

(6) log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modificaⁿ & stored on a stable storage media, which is failsafe.

(7) Log-based recovery works as follows:-

- The log file is kept on a stable storage media.
- When a transacⁿ enters the system & starts execⁿ, it writes a log about it.

<Tn, Start>

- When the transacⁿ modifies an item X, it writes logs as follows :-

<Tn, X, V1, V2>

It reads Tn has changed the value of X from V1 to V2.

- When the transacⁿ finishes, it logs —

<Tn, commit>

(8) The database can be modified using two approaches :-

(1.) deferred db modificaⁿ (2.) Immediate db modificaⁿ.

(1) Deferred db modificaⁿ: All logs are written on to the stable storage & the db is updated when a transacⁿ commits.

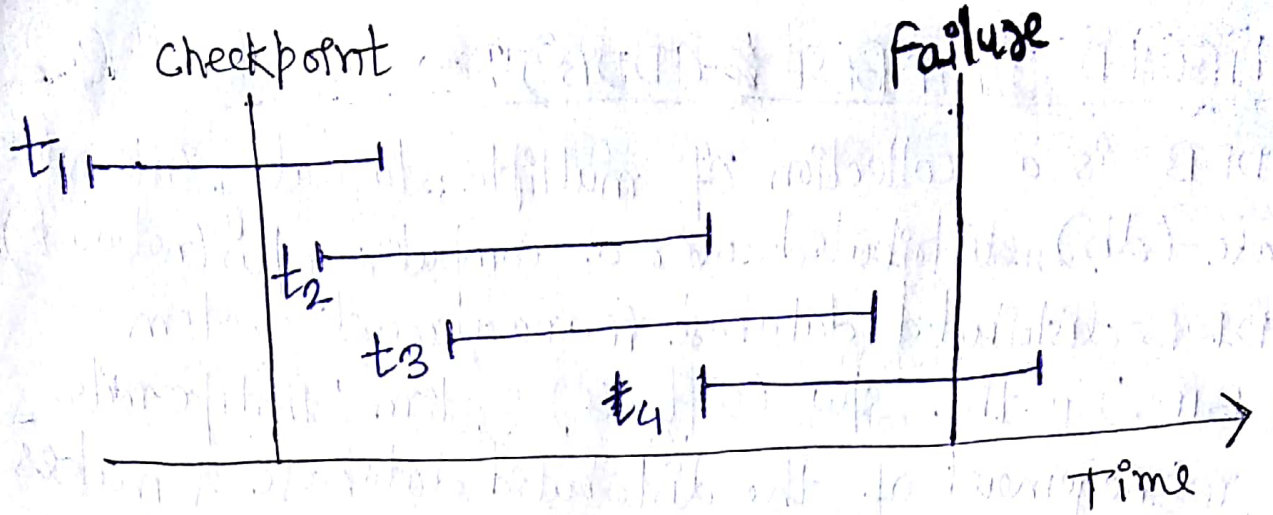
(2) Immediate db modificaⁿ: Each log follows an actual db modificaⁿ. That is, the db is modified immediately after every operation.

→ Checkpoints Technique to recover from ~~concurrent~~ transacⁿ failures

(1) It helps to overcome the limitaⁿ of log-based recovery.

(2) When more than one transacⁿ are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, & then start recovering. To ease this situation, most modern DBMS use the concept of "checkpoints".

(3) Checkpoint: Keeping & maintaining logs in real time & in real environment may fill out all the mem. space available in the sys. As time passes, the log file may grow too big to be handled at all.
Checkpoint is a mechanism where all the previous logs are removed from the system & stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, & all the transactions were committed.



By above figure;

(a) When a system with concurrent transac^{n}s crashes & recovers, it behaves in the following manner —

- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an Undo-list & a Redo-list.
- If the recovery system sees a log with <Tn, start> and <Tn, commit> or just <Tn, commit>, it puts the transaction in the Redo-list.
- If the recovery system sees a log with <Tn, start> but no commit or abort log found, it puts the transac^n in Undo-list.

(b) All the transac^{n}s in the Undo-list are then undone & their logs are removed.

(c) All the transac^{n}s in the Redo-list & their previous logs are removed & then redone before saving their logs.

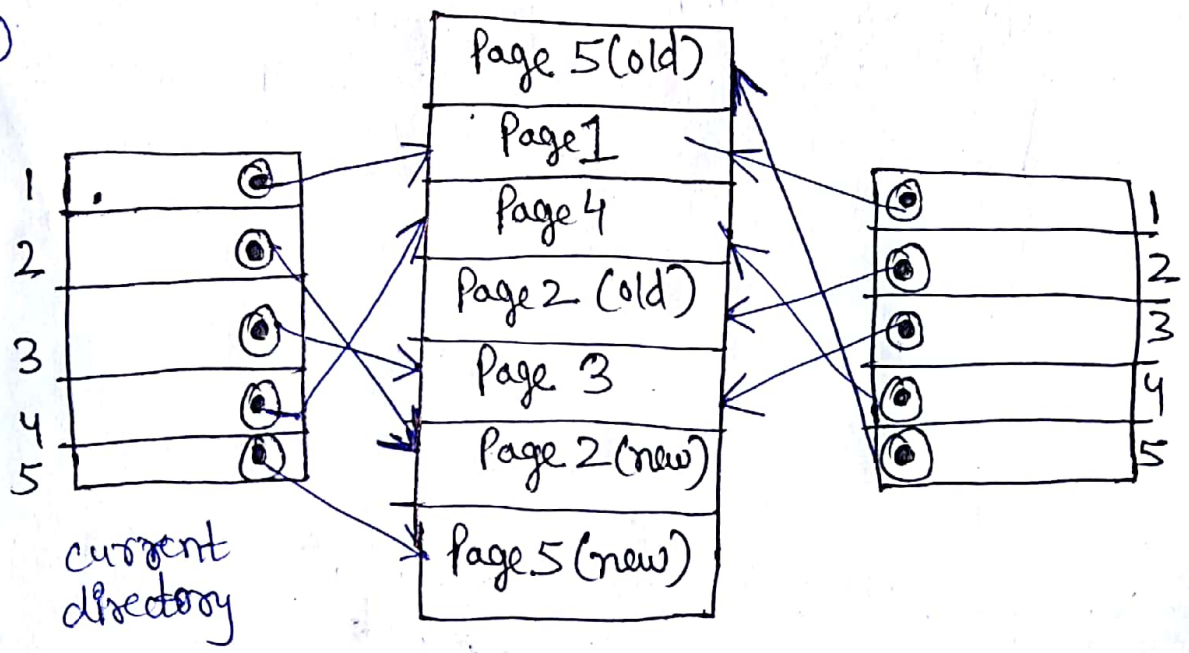
→ Backup Mechanism to recover from transac^n failure —

This mechanism is used to create backup copies of the db and the log files to be created at regular intervals w/o having to first stop the system. The backup copy of db can be used to recover the db in the event that db has been damaged or destroyed. A backup can be a complete copy of the entire db or an incremental copy. An incremental backup consists only of modifications made since the last complete or incremental backup. The backups usually stored on an offline-storage like magnetic tape.

→ SHADOW PAGING :-

- In a single user environment, shadow paging system can be used for data recovery instead of using transaction logs.
- In shadowpaging scheme, a db is divided into a number of fixed-sized disk pages, say 'n'. Therefore, a current directory is created that will be having 'n' entries with each entry pointing to a disk page in the db. The current directory is transferred to the main mem.
- When a transac^n begins executing, the current directory is copied into a shadow paging directory. The shadow directory is then saved on the disk. The transac^n will be using the current directory. Hence during the transac^n execution, all the modifica^n are made on the current directory and the shadow directory is never modified.

  *NOTE: When a transac^n performs the write operations, a new copy of the modified db page is created. This is done to ensure that the old copy of the db page is not overwritten until the transac^n commits.
- The current directory is modified to point to the new db page while the shadow directory continues to point to the old db page.
-



UNIT-5

Fig. illustrates the concepts of shadow paging. To recover from a failure occurring during the transacⁿ execution what we have to do is to just remove the new db pages as well as the current directory. The previous db version is safe in the old db pages pointed to by the entries in the shadow directory.

If the transacⁿ executes successfully, the current directory is copied to a single storage. The only thing we have to do is to change the pointer entries in the shadow paging directory which can be done by just overwriting the entries in shadow directory with the current directory.

*Note: Thus the shadow paging scheme reduces the need of transporting the updated db pages from the main memory to the disk storage.

for eg: In fig, the current directory is pointing to the pages 1, 2(new), 3, 4, 5(new). The new pages (2 & 5) are created as a result of changes made by the transacⁿ. Note, that the shadow directory points to the old pages (2 & 5) as well as the pages 1, 3 & 4 where there are no changes.

Besides, the main memory contains both the old & the new pages. Hence, if there occurs any failures, the old pages remain intact, pointed to by the shadow directory. If the transacⁿ commits successfully & the changes have to be written on to the stable storage, the contents of the shadow entry are overwritten by the contents of current directory. Also, the old pages (2 & 5) are removed from the memory and only the new modified pages are retained.

Concurrency Control :-
It is the process of managing simultaneous execution of transaction in a shared database, to ensure the ensure the serializability of transaction.

Purpose of Concurrency Control
i) To ensure isolation
ii) To preserve database consistency
iii) To resolve read-write & write-write conflict.

Concurrency Control Techniques
i) Lock-Based Protocol :-
A lock guaranties exclusive use of a data item to a current transaction.

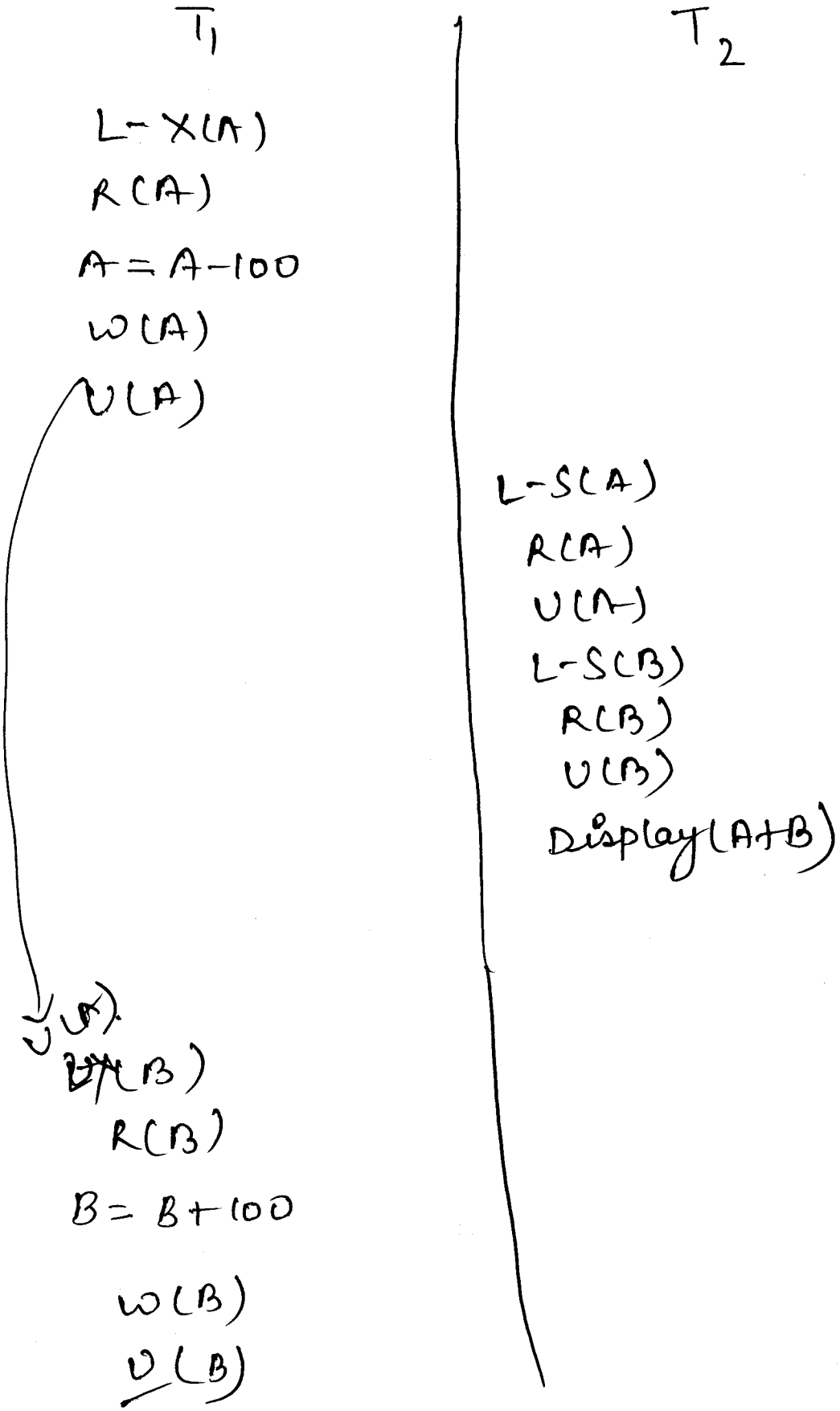
There are two modes of lock
i) shared lock (lock-S(Q))
ii) Exclusive lock (lock-X(Q))

	Shared	exclusive
Shared		
exclusive		

we need schedule must be consistent.

T_1	T_2
I_1	I_2

problem यह है कि same time पर write करना चाहते है। किसी तरह से isolation कर दो। अगर T_1 का I_1 काम कर रहा है तो T_2 का I_2 काम न करे। तो ऐसा होगा तो clash manage हो जाएगा।


T1
T2
L-X(A)
R(A)
A=A-100
W(A)
U(A)
L-S(A)
R(A)
U(A)
L-S(B)
R(B)
U(B)
Display(A+B)
R(B)
B=B+100
W(B)
U(B)

Concurrency Control Techniques

→ Till now already know how to check whether a schedule will maintain the consistency of DB or not.

→ Now will study protocols that guarantee to generate schedule which satisfy these properties specially (CS).

→ For Conflict serializability, we must avoid conflicting instructions we must remember three conditions of conflicting instruction

T_1	T_2
R(A)	
	W (A)

NOTE:- We cannot change the instructions of both transaction and will work on same data item and we can't change the transaction of each instruction but both instructions are working on same data item at the same time (we somehow manage that both the transaction will not access the data item at the same time)

→ Actual problem is different transactions trying to access data at same time.

→ How to approach conflict serializability

i) Time Stamping Protocol

ii) Lock - Based Protocol

ii) a) 2PL (Basic, Conservative, Strict, rigorous)

ii) b Graph- Based Protocol

iii) Validation Based Protocol

Lock- Based Protocol → (objective to make schedule consistent)

→ To achieve consistency, isolation is the most important idea, locking is simplest idea to achieve isolation. i.e first obtain a lock on a data item then performed a desired operation and then unlock it.

Note (means one transaction will work on one data item so we provide lock to that data item and other transaction must have to wait until the lock is released by the other transaction but in this scenario we are not able to achieve Concurrency)

→ To provide better concurrency along with isolation we use different modes of lock

T_1	T_2
lock-S(Q)	
	lock-S(Q)

Shared mode : → denoted by lock-S(Q)

Transaction can perform read operation any other transaction can also obtain same lock on same data item.

Exclusive mode → denoted by lock-X(Q).

Transaction can perform read and write operations, any other transaction cannot obtained either shared / Exclusive mode lock.

T_1	
lock-X(Q)	

Compatability table

	Shared	Exclusive
Shared	T	F
Exclusive	F	F

__/__/____

Note → Now question arises that if we do all above things, ~~that~~ we guarantee that the schedule is CS, VS, Recoverability, Cascadelessness, no deadlock will occur.

TWO-PHASE LOCKING PROTOCOL

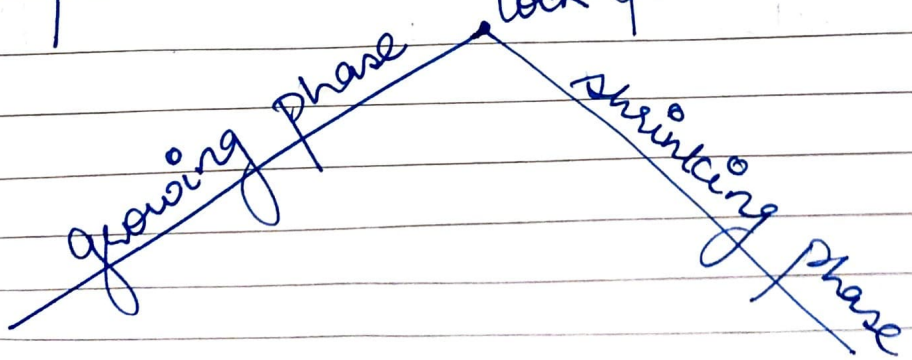
2PL / Basic 2PL

→ This protocol requires that each transaction in a schedule will be two phased Growing phase and shrinking phase.

→ In growing phase transaction can only obtain locks but cannot release any lock.

→ In shrinking phase transaction can only release locks but cannot obtain any lock.

→ Transaction can perform read/write operation both in growing/shrinking phase.



__/__/____

Ex- T_1

Growing
- lock-S(A)
- Read(A)
- lock-S(B)
- Read(B)

Shrinking
- unlock(A)
- unlock(B)

Q- S_1

T_1	T_2
lock-X(A)	
R(A)	
W(A)	
	lock-X(B)
	W(B)
	Unlock(B)
Unlock(A)	

↑ (2-Phase Locking)

Q- S_2

T_1	T_2
LOCK-S(A)	
R(A)	
Unlock(A)	
	LOCK-X(B)
	W(B)
	Unlock(B)

↑ Not in 2-Phase

→ Ensure conflict serializability & view serializability is the order in which transaction reaches to the lock point.

→ May generate recovarable schedules and cascading rollbacks.

S

T_1	T_2
lock-X(A)	
R(A)	
W(A)	
Unlock(A)	
	lock-S(A)
	R(A)

Dirty read

Note (If there is a dirty read, then definitely there is a cascading rollbacks).

ii) If Transaction T_2 commited first before transaction T_1 so it is a irrecoverable schedule).

ii) explanation

S

T_1	T_2
lock-X(A)	
R(A)	
W(A)	
Unlock-(A)	
	lock-S(A)
	R(A)
	Unlock(A)
	Commit
Commit	

dirty read

→ Do not ensure freedom from deadlock.

T_1	T_2
lock-S(A)	
Read(A)	
	lock-X(B)
	Read(B)
	write(B)
lock(B)	
	lock-X(A)

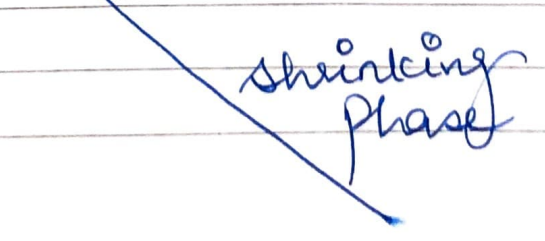
deadlock state

Conservative 2PL / Static 2PL

→ There is no growing phase transaction first will acquire all the lock required and then directly will start from lock-point.

→ If all the locks are not available then transaction must release the lock acquired so far and wait.

→ Shrinking phase works as usual and transaction can unlock any data item anytime



__/__/____

→ Here we must have all the knowledge that what data items will be required during execution.

→ Here we achieve Conflict serializability, view serializability and free from deadlock.

→ Possibility of irrecoverable schedules and cascading rollbacks.

T_1	T_2
lock-X(A)	
R(A)	
W(A)	dirty read
Unlock(A)	
	lock-S(A)
	R(A)

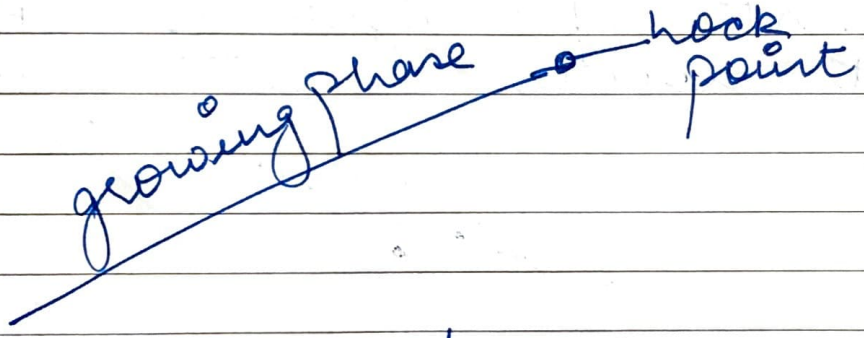
suppose T_1 want only work on one data item. so it lock it and unlock it.

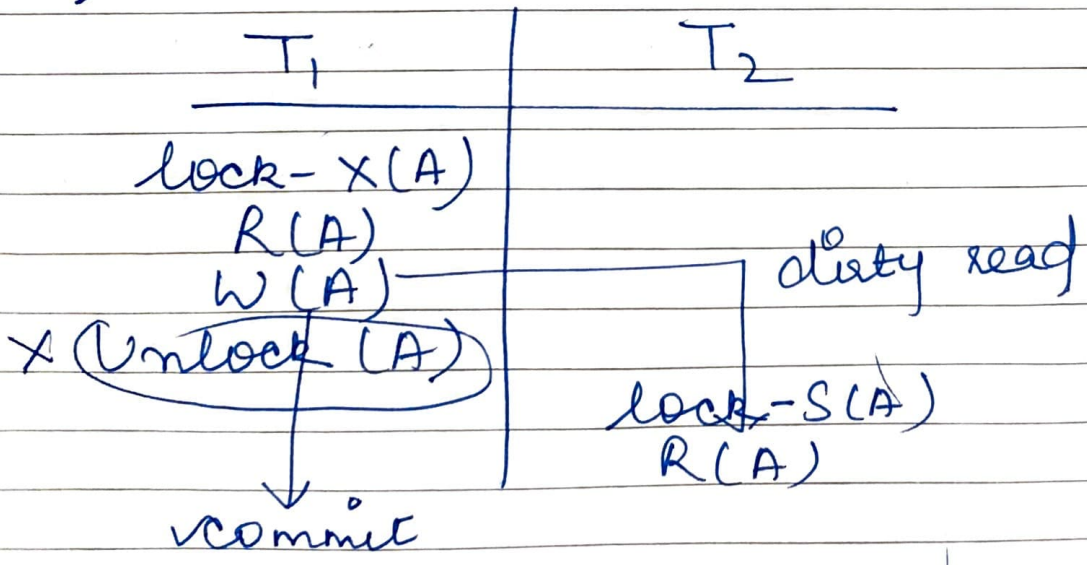
→ So in this protocol, irrecoverability and cascadeleness problem exist.

Rigorous 2PL ↓

→ It is an improvement of 2PL protocol where we try to ensure recoverability and cascadelessness.

→ Rigorous 2PL requires that all the locks must be held until transaction consists i.e. there is no shrinking phase in the life of transaction.





→ Here we cannot unlock any transaction and if we don't do unlocking then different transaction cannot read the value and if different transaction cannot read the value

then dirty read problem will not exist. If there is no dirty read then schedule will be recoverable and there is no cascading rollbacks.

→ Suffer from deadlock and inefficiency.

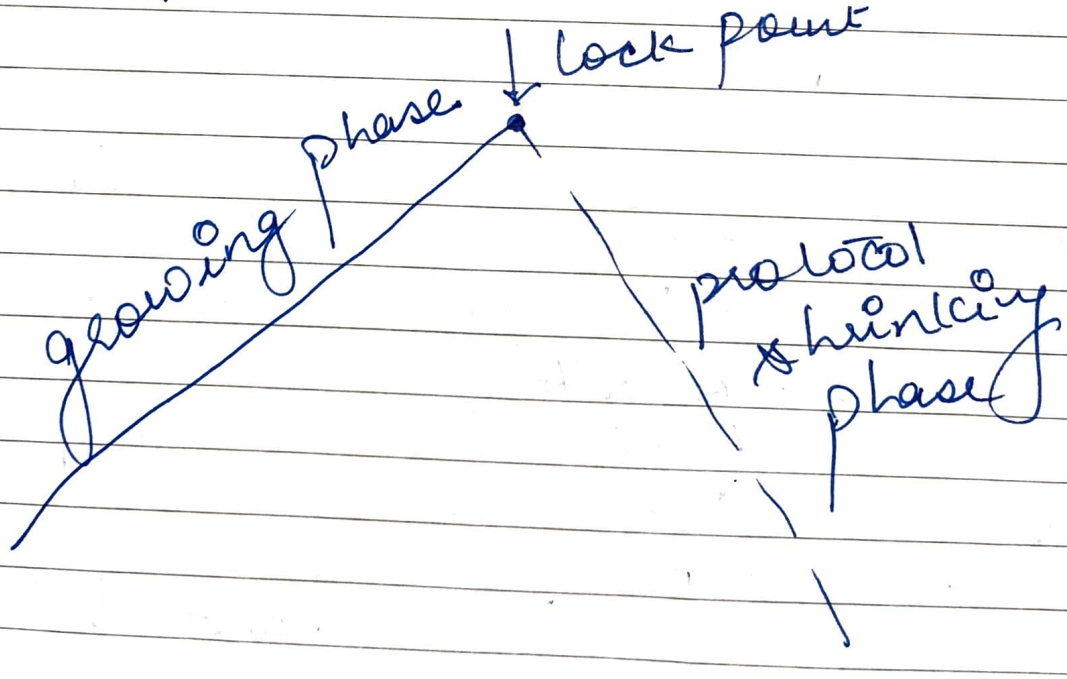
Note:- You are saying that if you lock any data item then you will not unlock it. If you will not unlock any data item then no other transaction can lock the data item. So here we cannot achieve concurrency).

Strict 2PL

→ It is an improvement over Rigorous 2PL.

→ In the shrinking phase unlocking of Exclusive lock are not allowed but unlocking of shared locks can be done.

→ all the properties are same as that of rigorous 2PL, but it provide better concurrency.



Graph Based Protocol

→ If we wish to develop lock based protocol that are not based on 2PL we need additional information that how each transaction will access the data.

→ There are various model that can give additional information each differing in the amount of information provided.

→ One idea is to have prior knowledge about the order in which the database items will be accessed.

→ We improve partial ordering → on set all data items $D = \{d_1, d_2, d_3 \ldots d_n\}$ if $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access d_i before d_j.

→ Partial ordering may be because logical or physical organisation or only because of concurrency control.

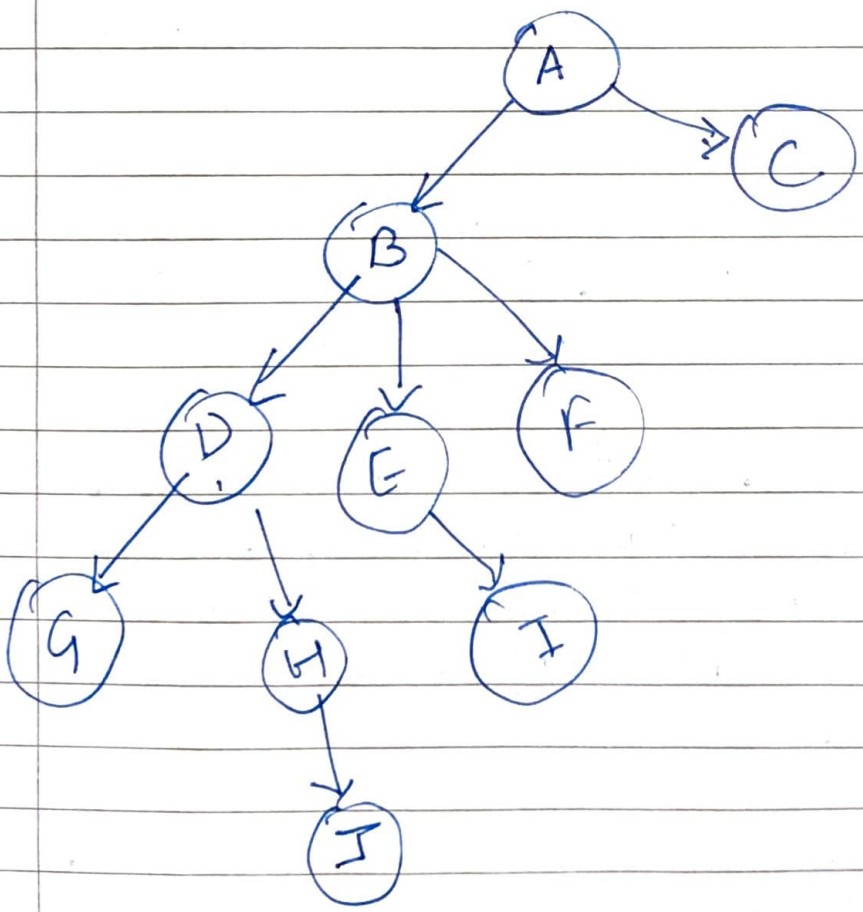
Note- { logical means link list
physical means memory will be accessed sequentially }

→ After partial ordering, set of all data items D will know be viewed as directed acyclic graph called database graph.

→ For sake of simplicity, we follow two restrictions –

i) Will study graphs that are rooted tree.

ii) Will use only exclusive mode locks.



__/__/____

Tree Protocol

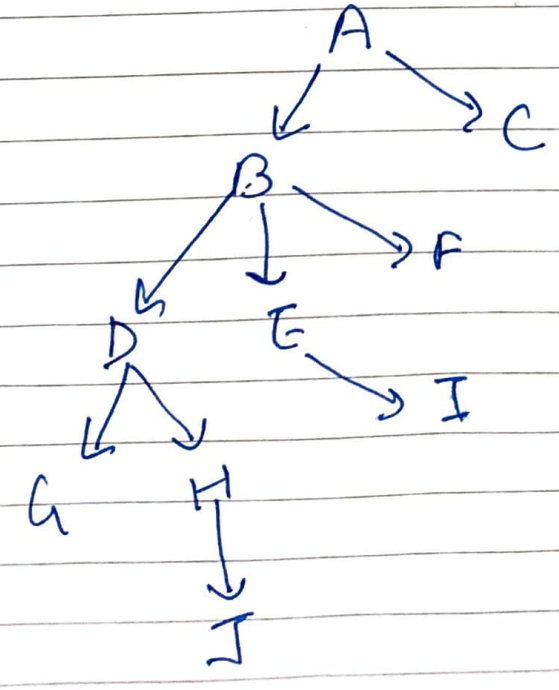
→ Each Transaction T_i can lock a data item Q with following rules—

i) First lock by T_i may be on any data item.

ii) Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.

iii) Data item may be unlocked at any time.

iv) Data item Q that has been locked and unlocked by T_i cannot subsequently by relocked by T_i.



T_1	T_2
lock X(B)	lock(A)
lock X(D)	lock(B)
lock X(E)	lock(C)
Unlock X(B)	Unlock(A)
Unlock(D)	lock(D)
Unlock(E)	Unlock(B)
	lock(G)
	Unlock(D)
	Unlock(G)
	Unlock(C)

Note - In the transaction T_2 only it wants to work on ~~data item~~ G & C but we can not directly G and C simultaneously so we have to find common point where we can lock the item simultaneously.

__/__/____

Properties of Tree based Protocol

i) All schedules that are legal under the tree protocol are conflict serializable and view serializable.

ii) Tree protocol ensure freedom from deadlock.

iii) Tree protocol do not sure for recoverability and cascadelessness.

iv) Early unlocking is possible which leads shortest waiting time an increase concurrency.

v) A transaction may have to lock data item that it does not access lead to overhead, waiting time and decrease in concurrency.

→ Transaction must know exactly what data item are to be accessed.

→ Recoverability and Cascadeleness can be provided by not unlocking before Commit.

Deadlock handling

→ A system is in deadlock state if there exists a set of transaction such that every transaction in the set is waiting for another transaction in the set.

→ If there exists a set of waiting transactions $T_0, T_1 - - - T_{n-1}$ such that $T_0 \rightarrow T_1$, $T_1 \rightarrow T_2$, $T_2 - - - T_{n-1}$, $T_{n-1} \rightarrow T_0$. So none transaction can progress in such situation.

→ System must have a proper methods to deal with deadlocks, otherwise

i) in real time system, it may lead to life and money.

ii) will reduce resource utilization and increase inefficiency.

There are too principles for dealing with deadlock problem.

a) Deadlock Prevention → which ensure that system will never enter a deadlock state.

detection &

b) Deadlock Recovery → allow system to enter into deadlock, then try to recover.

→ Graph-based Protocol, Conservative 2PL Time Stamping Protocol, Tree-based Protocol. Then is no possibility of deadlock.

→ When risk is higher then use deadlock prevention.

→ When there is a low risk then use deadlock detection and recovery.

Conditions for Deadlock

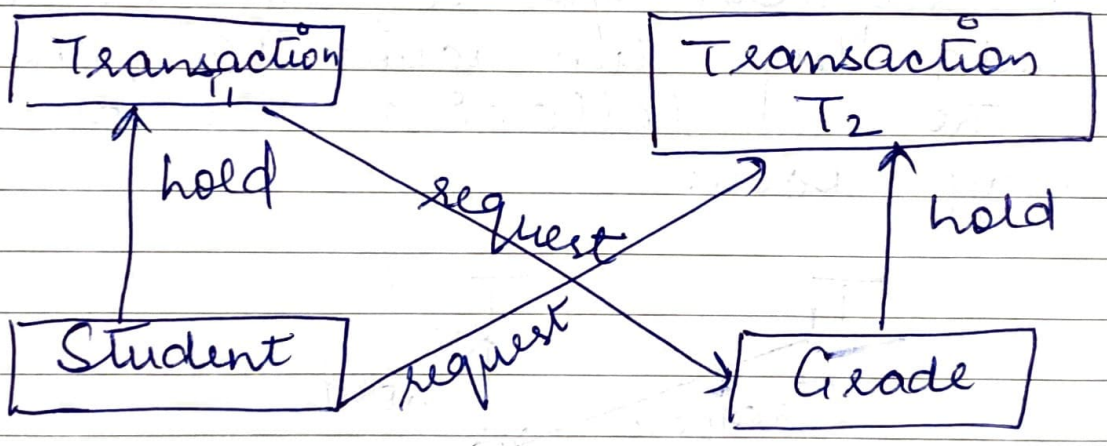
i) Mutual Exclusion ii) hold and wait
iii) No Preemption iv) Circular wait.

example to understand deadlock ↓

het us understand the concept of deadlock with an example —

Suppose, Transaction T1 holds a lock on some rows in the Student table and needs to update some rows in the Grade Tables. Simultaneously, Transaction T2 holds locks on those very rows (which T1 needs to update) in the Grade Table but needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Transaction T1 will wait for Transaction T2 to give up lock and similarly, Transaction T_2 will wait for transaction T1 to give up the lock. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock Prevention

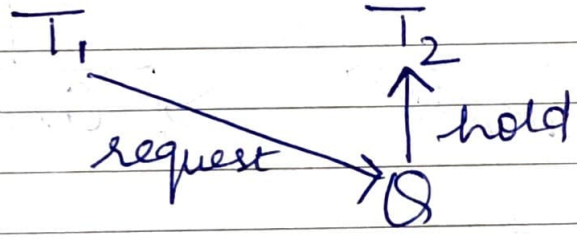
→ For large databases, deadlock prevention method is suitable. A deadlock can be prevented if the resources allocated in such a way that deadlock never occur.

→ The DBMS analyzes the operations whether they can create deadlock situation or not. If they do, that Transaction is never allowed to be executed.

There are some deadlock prevention schemes that use timestamps in order to make sure that deadlock does not occur. These are —

i) Wait - Die Scheme ↓

In the wait-die scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two possibilities may occur.

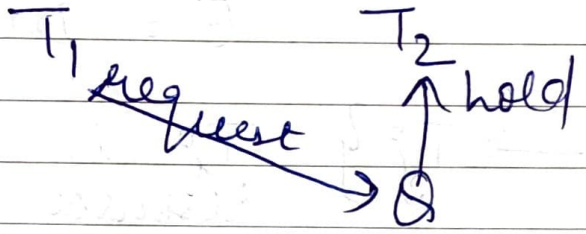


i) TS(T1) < TS(T2) → If T1 is older than T2 i.e. T1 came in the system earlier than T2, then it is allowed to wait for the resource which will be free, when T2 has completed its execution.

ii) TS(T1) > TS(T2) → If T1 is younger than T2 i.e. T1 came in the system after T2, then T1 is killed. It is restarted later with the same timestamp.

Wound-Wait Scheme ↓

In the wound-wait scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two possibilites may occur.



i) If TS(T1) < TS(T2) – If T1 is older than T2 i.e T1 came in the system earlier than T2, then it is allowed to rollback T2 or wound T2. Then T1 takes the resource and completes its execution T2 is later restarted with the same timestamp.

ii) TS(T1) > TS(T2) → If T2 is older than T1 i.e. T2 came in the system earlier than T1, then T1 is allowed to wait for the resource which will be free when T2 has completed its execution.

__/__/____

Deadlock Detection

Simple way to detect a state of deadlock is to draw wait-for graph.

$G = (V, E)$ V = Nodes describing transactions
E = Directed edges

$\{T_i \rightarrow T_j\}$ T_i is waiting for a resource data item held by T_j.

→ When a transaction T_i requests a data items currently being held by transaction T_j then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph.

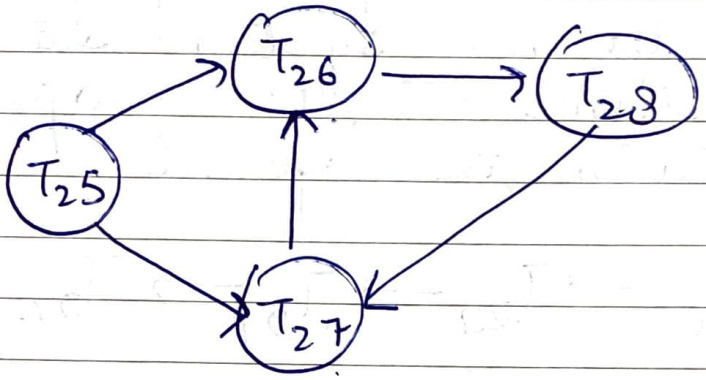
→ This edge is removed only when transaction T_j is no longer holding a data item needed by Transaction T_i.

→ A deadlock exists in the system if and only if the wait-for graph contains a cycle.

→ Each transaction involved in the cycle is said to be deadlocked.

→ To detect deadlocks, the system needs to maintain the wait-for graph and

and periodically to invoke an algorithm that searches for a cycle in the graph.



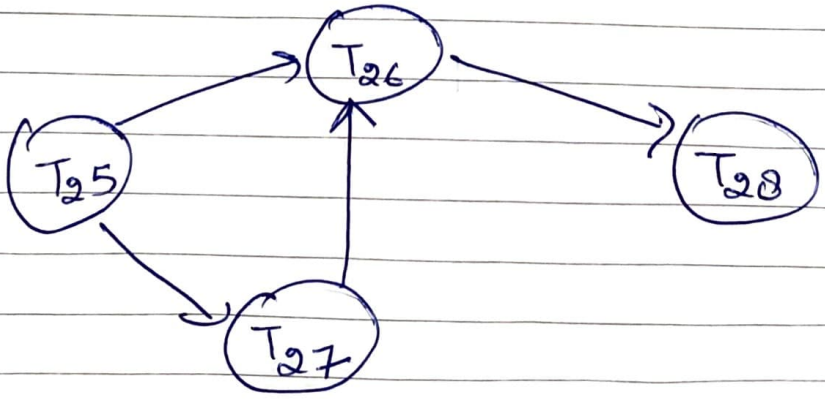
Wait-for graph with a cycle

Illustrate the concept —

i) Transaction T_{25} is waiting for transaction T_{26} and T_{27}.

ii) Transaction T_{27} is waiting for transaction T_{26}.

iii) Transaction T_{26} is waiting for transaction T_{28}.



Since graph has no cycle, the system is not in a deadlock state.

→ Suppose now that transaction T_{28} is requesting an item held by T_{27}. The edge $T_{28} \rightarrow T_{27}$ is added to the wait-for graph, resulting in the new system state. ~~In~~ fig 2. This time, the graph contains the cycle

$$T_{26} \rightarrow T_{28} \rightarrow T_{27} \rightarrow T_{26}$$

implying that transactions T_{26}, T_{27} and T_{28} all deadlocked.