→ Describe how a non-recoverable situation could arise if write locks are released after the last operation of a transaction but before its commitment.

The DLM uses a generalized concept of resources which is some entity to which shared access must be controlled. This may relate to file a record or an area of shared memory but can be anything that the application designer chooses. A hierarchy of resources may be defined so that no. of levels of locking can be implemented. for instance, a hypothetical database might define a resource hierarchy as follows -:

Database
↓
Table
↓
Record
↓
file

A process can then acquire locks on the database as a whole & then on particular parts of the database. A lock must be obtained on a parent resource before a subordinate resource co

be looked.

→ Lock Modes

A process running within a VMS cluster may obtain a lock on a resource. There are six lock modes that can be granted, & these determine the level of exclusivity of access to the resou Once a lock has been granted it is possible to convert the lock to a higher or lower levels of lock mode. When all process have unlock a resource, the system's information about the resource is destroyed.

→ Null Lock

Indicates interest in the resource but does not prevent other processes from locking it. It has the advantage that the resource & its lock value blocks are preserved even when no proce are locking it.

→ Concurrent Read (CR)

Indicates a desire to read (but not update) th resource. It allows other processes to read or update the resource but prevents other from having exclusive access to it. This is usually employed on high-level resources in order th higher levels of lock can be obtained on subo resources.

→ Concurrent Write (CW).

Indicate a desire to read lock, which indicate a desire to read the resource. It also allows other processes to read or update the resource, but prevent others from having exclusive access to it. This is usually employed on high-level resources in order that higher levels of locks can be obtained on subordinate resources.

→ Protected Read (PR)

This is a traditional update lock, which indicates a desire to read & update the resource & prevent others from updating it. Others with Concurrent read access can however read the resource.

→ Exclusive

Traditional exclusive lock that allows read & update access to the resource & prevent other from having any access to it.

→ Protected Write (PW).

This is a traditional update lock which indicates a desire to read & update the resource & prevent others from updating it. Others with Concurrent read access can however read the resources.

An earlier transaction may release its locks but not commit, meanwhile a later transaction uses the objects & commit. The the earlier transaction has done a dirty read & cannot be recovered because it has already committed.

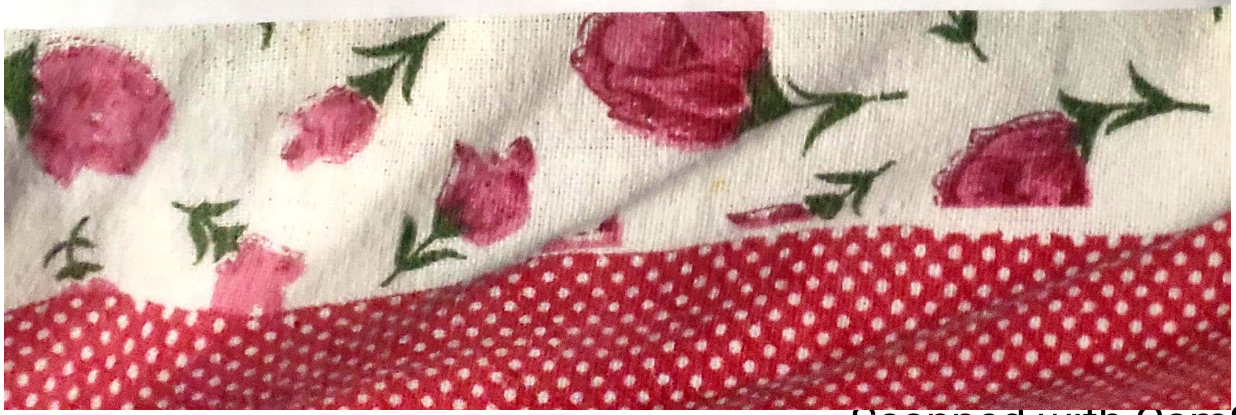
It basically means that if the write lock are released before the data is saved, then a new transaction might come, use the data, change the value & commit before the commit of the first transaction occurred. After that at some time later, the 1st transaction finishes its commit.

There are 2 issues here, the data read by the second transaction is old data (in ideal situation it should have used the value of which first transaction provides) & the other issue that if that after the 1st transaction finishes its commit, the commit done by second transaction is lost.

let us first understand the concept of recaverable & non-recoverable schedules -:

(i) Recoverable Schedule -:

In which transaction commit only after all transactions whose changes they read commit are recoverable schedule.



Exam →

T_1	T_2
R(A)	
W(A)	
	W(A)
	R(A)
Commit.	

T_1 → is executed before T_2 & hence no chance of conflict occur. $R_1(x)$ appear before $W_1(x)$ & transaction T_L committed before T_2.

(ii) Irrecoverable Schedules

T_1	T_1's buffer space	T_2	T_2's buffer space	Database
				A = 5000
R(A);	A = 5000			A = 5000
A = A − 1000;	A = 4000			A = 5000
W(A);	A = 4000.			A = 4000
		R(A);	A = 4000	A = 4000
		A = A + 500;	A = 4500	A = 4000
		W(A);	A = 4500	A = 4500
		Commit;		
Failure point				
Commit;				

T_L reads & writes A & that value is read & written by T_2. T_2 commits, but later T_1 fails So we have to rollback T_L. Since T_2 has read the

value written by T_1, it should be rolled back but we have already committed that. So, this is an irrecoverable schedule.

Now, if a write lock is released after the last operation of transaction but before the commitment a consider arise like —:

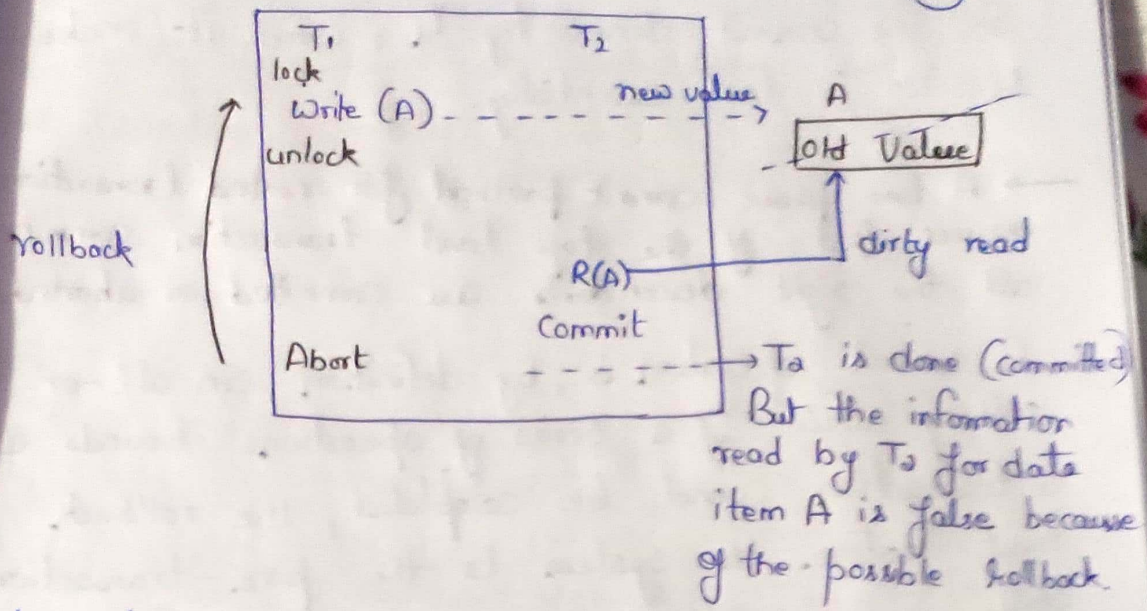
Suppose we have M = 500 ₹ in the bank,

T_1 adds ₹ 100. It reads the value of 500 & changes it to 600. However, it release the lock before commiting these values. Interrupt occurs & this transaction goes to sleep.

T_2 also wants to add 100. to the account. It reads the value of M, which is 500. (as T_1 did not commit this). change this to 600 & then commit. Now we have 600 in our acco

Then T_1 wakes up & finishes its commit, i.e. committing the value of M=600 as it is unaw of the transaction T_2.

So, final amount in the account after both commits, M=600, even when 2 deposits of 100 were made & this should have been 700.



Now, the other issue is —:

Transaction T_1 does not anything about T_2. Let, T_1 & T_2 both want to add 10 & 20 to the variable, M whose current value is 100. T_1 comes first, reads M = 100. adds 10, makes it 110 in its own local copy, before commit, it goes to sleep.

T_2 comes; reads M = 100, add 20, makes its 120 and commits, M is now 120.

T_1 wakes up; & finishes its commit by saving its local M value (110). After commit M = 110.

So, the commit done by T_2, when it makes $M = N_0$ is lost essentially.

→ A two phase commit protocol for nested transaction ensures that if the top level transaction commits, all the right descendants are committed or aborted.

A commit operation is, by definition, an all-or-nothing affair. If a series of operations bounds as a transaction cannot be completed, the rollback must restore the system to the pre-transaction state.

In order to ensure that a transaction can be rolled back, a software system typically logs each operation, including the commit operation itself. A transaction/recovery manager uses the log records to undo a partially completed transaction.

When a transaction involves multiple distributed resources the commit process is somewhat comp because the transaction includes operations that span two distinct software systems, each with its own resource manager, log records & so on. In this case, the distributed resources are the nes transaction. But when the top level transaction commits all the rights of descendents are commi

→ Example of interleaving of 2 transactions that is serially equivalent at each server but is not serially equivalent globally.

T	U
Read (A) at x	Read (B) at y
Write (B)	Write (B)
Read (B) at y	Read (A) at x
Write (B)	Write (A)

x & y are 2 different servers with T & U as transactions.

Schedule at server X

T: Read (A); Write (A);

U: Read (A); Write (A);

Seriall equivalent with T before U.

Schedule at Server y

U: Read (B); Write (B);

T: Read (B); Write (B);

Seriall equivalent with U before T.

This is not serially ~~equiall~~ equivalent globally because there is a cycle T → U → T.

Explain why executions are always strict.
if read lock are released after the last
eration of a transaction but before its committment.

guarantee serializability, we must follow some
itional protocol, concerning the position of locking
unlocking operations in every transaction.
o phase locking protocol (2-PL) ensures serializability.
Growing Phase
New locks on data items may be acquired but
one can be released.
Shrinking Phase
xisting locks may be released but no new
ocks can be acquired.
→ If lock conversion is allowed, then upgrading
f lock (from S(a) to X(a)) is allowed in
rowing Phase & downgrading of lock must be
one in shrinking phase.

	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	- - -	- - -
5	Unlock (A)	lock-X(C)
6		
7	Unlock (B)	Unlock (A)
8		
9		Unlock (C)
10	- - -	- - -

For T_1 -: Growing phase is from steps 1 - 3
Shrinking phase is from steps 5 - 7
Lock point at 3.

For T_2 -: Growing phase is from steps 2 - 6
Shrinking phase is from steps 8 - 9
Lock point at 6.

Lock Point → A point at which the growing phase ends,

Categories of 2 - PL.

(i) Strict 2 - PL (ii) Rigorous 2 - PL
(iii) Conservative 2 - PL

Strict 2 - PL

This requires that in addition to the lock being 2 - phase all Exclusive (x) locks held by the transaction be released until after the transaction commits. 2PL ensures a recoverable & cascadeless schedule.

It gives us freedom from Cascading Abort which was there in 2 - PL & gurantee strict schedule, but still deadlocks are possible.

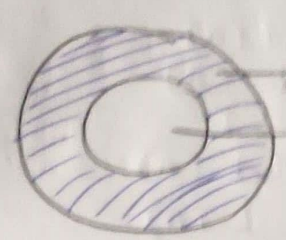
Rigorous 2 - PL

Requires that in addition to the lock being 2 - Phase all Exclusive (x) & Shared (s

locks held by the transaction can be released after the transaction commits.

Conservative 2-PL

It requires the transaction to lock all the items access before the transaction begins execution by predeclaring its read-set & write set.



Example →

	T_1	T_2
1	Read (A)	
2		Read (A)
3	Read (B)	
4	Write (B)	
5	Commit	
6		Read (B)
7		Write (B)
8		Commit.

The schedule here is conflict serializable, so →:

	T_1	T_2
1	Lock-S(A)	
2	Read (A)	
3		Lock-S(A)
4		Read (A)

5	Lock -X (B)	
6	Read (B)	
7	Write (B)	
8	Commit	
9	Unlock (A)	
10	Unlock (B)	
11		Lock -X (B)
12		Read (B)
13		Write (B)
14		Commit
15		Unlock (A)
16		Unlock (B).

Shared & Exclusive Locks (Strict 2-PL).

T_1	T_2.
read_lock (Y)	
read_item (Y)	
write_lock (X)	
	read_lock (X)
read_item (X)	
X = X+Y	
write_item (X)	
Commit	
unlock (Y), unlock (X)	
	read_item (X)
	write_lock (Y)
	read_item (Y)
	Y := X+Y
	write_item (Y)
	Commit
	unlock (X), unlock (Y)