Recoverability

Recoverability is a property of database systems that ensures that, in the event of a failure or error, the system can recover the database to a consistent state. Recoverability guarantees that all committed transactions are durable and that their effects are permanently stored in the database, while the effects of uncommitted transactions are undone to maintain data consistency.

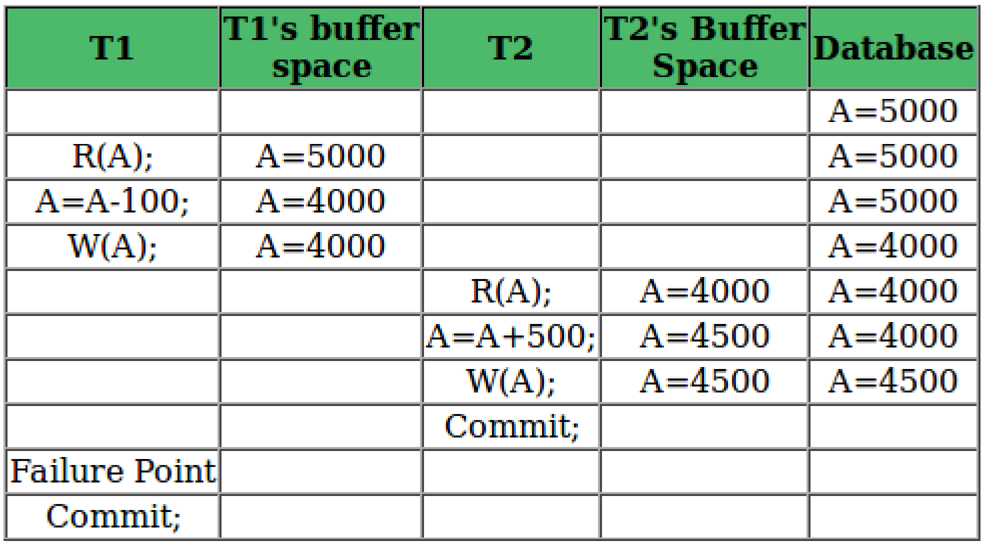
Recoverable Schedules: Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	**T_2**
R(A)	
W(A)	
	W(A)
	R(A)
commit	
	commit

This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.

Irrecoverable Schedule: The table below shows a schedule with two transactions, T1 reads and writes A and that value is read and written by T2. T2 commits. But later on, T1 fails. So we have to rollback T1. Since T2 has read the value written by T1, it should also be rollbacked. But we have already committed that. So this schedule is irrecoverable schedule. When Tj is reading the value updated by Ti and Tj is committed before committing of Ti, the schedule will be irrecoverable.



T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	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;				

Recoverable with Cascading Rollback: The table below shows a schedule with two transactions, T1 reads and writes A and that value is read and written by T2. But later on, T1 fails. So we have to rollback T1. Since T2 has read the value written by T1, it should also be rollbacked. As it has not committed, we can rollback T2 as well. So it is recoverable with cascading rollback. Therefore, if Tj is reading value updated by Ti and commit of Tj is delayed till commit of Ti, the schedule is called recoverable with cascading rollback.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	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
Failure Point				
Commit;				
		Commit;		

Cascadeless Recoverable Rollback: The table below shows a schedule with two transactions, T1 reads and writes A and commits and that value is read by T2. But if T1 fails before commit, no other transaction has read its value, so there is no need to rollback other transaction. So this is a Cascadeless recoverable schedule. So, if Tj reads value updated by Ti only after Ti is committed, the schedule will be cascadeless recoverable.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
Commit;				
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		

Concurrency Control Protocols: The concurrency control protocols ensure the atomicity, consistency, isolation, durability and serializability of the concurrent execution of the database transactions. Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. These protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

Lock Based Protocols – 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. They synchronize the access by concurrent transactions to the database items. It is required in this protocol that all the data items must be accessed in a mutually exclusive manner. 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):** also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.
2. **Exclusive Lock (X):** Data item can be both read as well as written. This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

	S	X
S	√	≠
X	≠	≠

There are four types of lock protocols available:

1. 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.

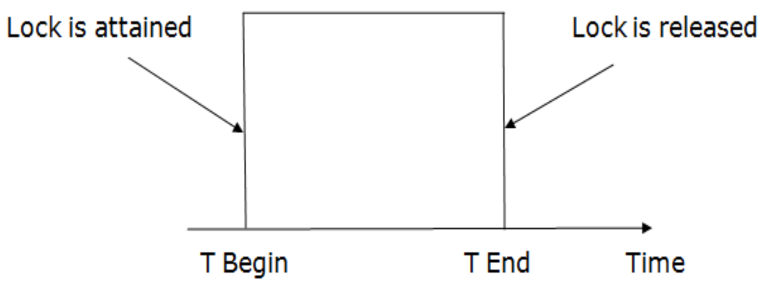
S. NO.	T1	T2
	Lock X(A)	
	Read(A)	
	Write(A)	

	Unlock(A)	
		Lock S(A)
		Read(A)
	Lock X(B)	
	Read(B)	
	Write(B)	
	Unlock(B)	

May not produce only serializable schedule. Irrecoverability, and deadlock.

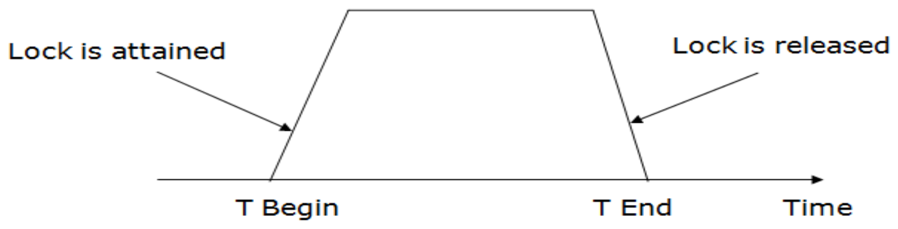
2. Pre-claiming Lock Protocol

- 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.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:
Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.
Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

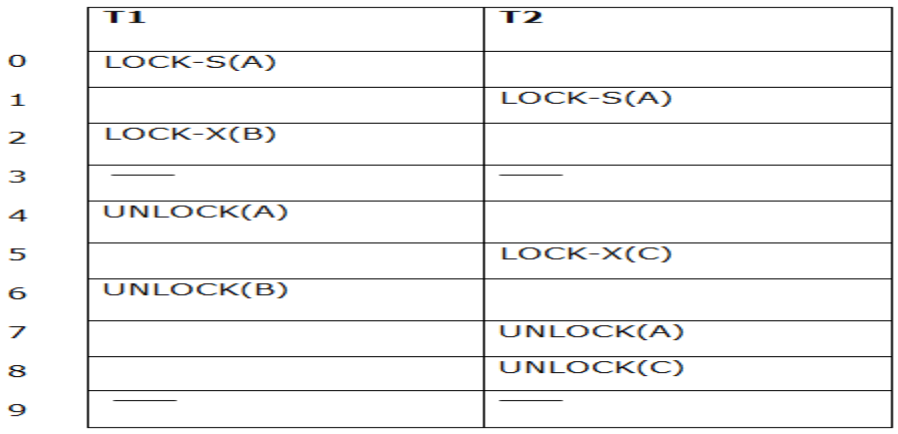
1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

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



	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

2-PL ensures serializability, but there are still some drawbacks of 2-PL.

- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation are possible.