Concurrency Control

- In a multiprogramming environment where multiple transactions can be executed simultaneously, controlling the concurrency of transactions is highly important.
- We have concurrency control protocols to ensure concurrent transactions' atomicity, isolation, and serializability.
- Concurrency control protocols can be broadly divided into two categories −

  – Lock based protocols

  – Time stamp-based protocols

Concurrency Control

- Locking a data item being used by a transaction can prevent other transactions running simultaneously from using these locked data items.

The following rules must be followed whenever binary locking schemes are used:

- **Lock ():** This operation must be issued by the transaction before any update operations such as a read or write performed on the transaction.
- **Unlock ():** This operation must be issued by the transaction after all read or write operations in the transaction have completed.

T1	T2	Schedule
Lock (A)	Lock (A)	T1: Lock (A)
Read (A)	Read (A)	T1: Read (A)
A: = A + 200	A: = A + 300	T1: A: = A + 200
Write (A)	Write (A)	T1: Write (A)
Unlock (A)	Unlock (A)	T1: Unlock (A)
		T2: Lock (A)
		T2: Read (A)
		T2: A: = A + 300
		T2: Write (A)
		T2: Unlock (A)

Lock-based Protocols

- 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 locks available **Shared S(a)** and **Exclusive X(a)**.

1. Shared lock:

- It is also known as a **Read-only lock.** In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both read as well as written by the transaction.
- This lock is exclusive, and in this lock, **multiple transactions do not modify the same data simultaneously.**

Shared and Exclusive lock

Transaction T1	
LOCK S (A)	// Shared Lock on data "A"
R (A)	// Read Operation on data "A"
Unlock (A)	// Unlock data "A"

Transaction T1	
LOCK S (A)	// Exclusive Lock on data "A"
R (A)	// Read Operation on data "A"
W (A)	// Write Operation on data "A"
Unlock (A)	// Unlock data "A"

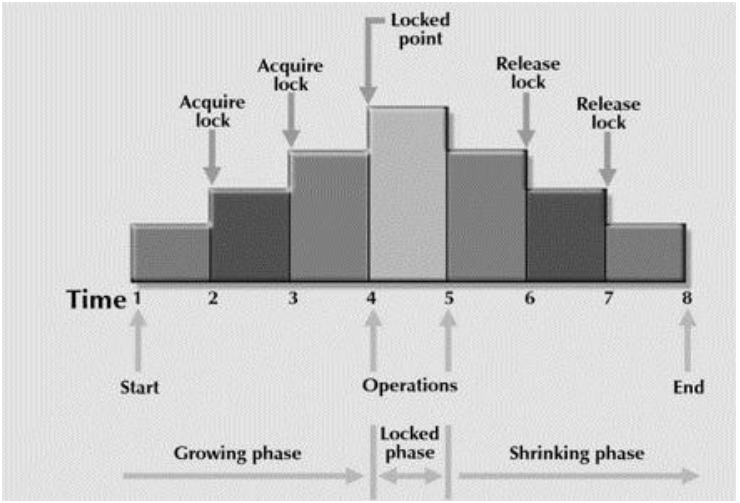
Locking protocols

Two-Phase Locking 2PL:

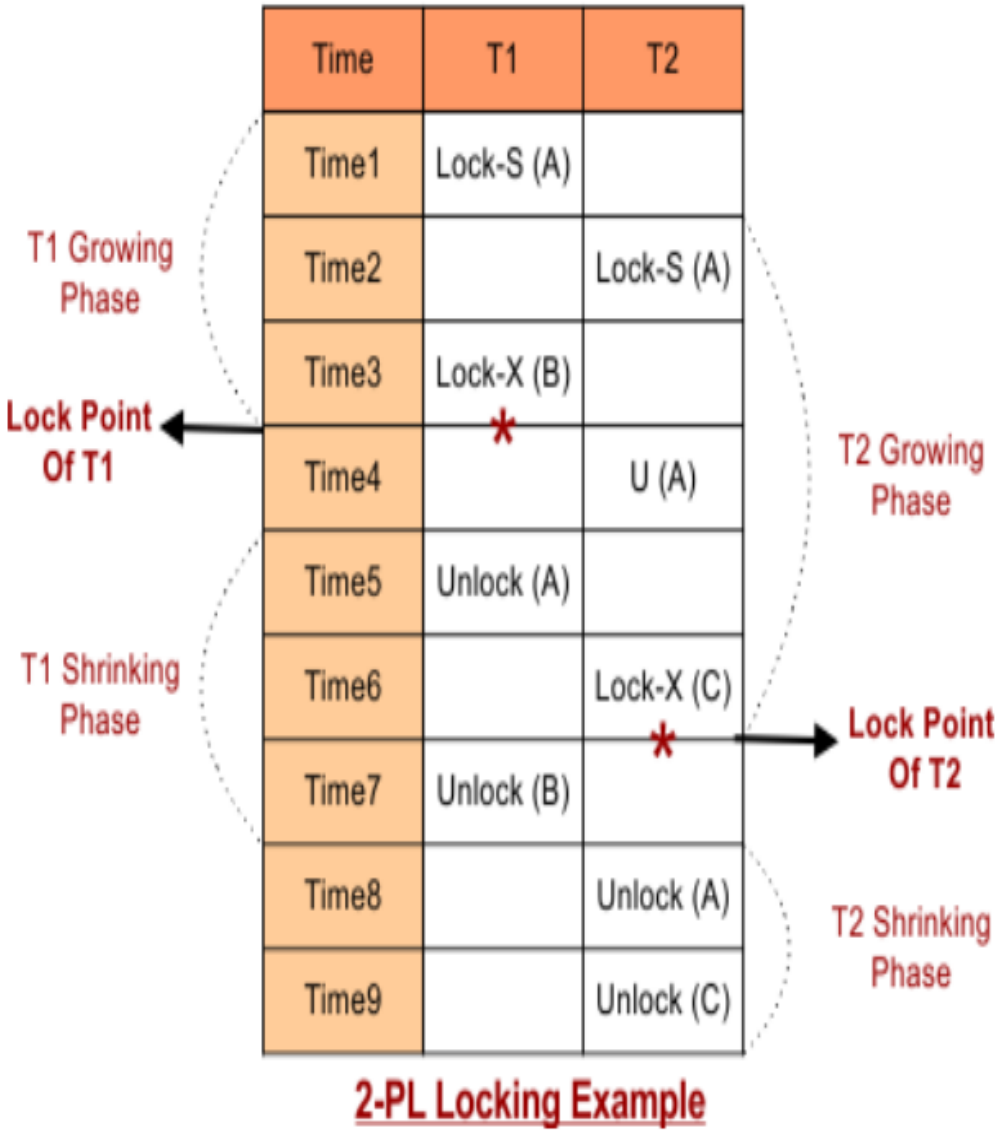
- A transaction is said to follow the Two-Phase Locking protocol if locking and unlocking can be done in two phases.

  1. **Growing Phase**: In the Growing phase, only Locks are acquired by a transaction and no locks are released by a transaction at that time.

  2. **Shrinking Phase**: In the shrinking phase, only Locks are released by transaction, and no locks are acquired by a transaction at that time.



Let's see a transaction implementing 2-PL.



2-PL Locking Example

Transaction T1

- **Growing Phase:** From Time 1 to 3.
- **Shrinking Phase:** From Time 5 to 7
- **Lock Point:** At time 3

Transaction T2

- **Growing Phase:** From Time 2 to 6
- **Shrinking Phase:** From Time 8 to 9
- **Lock Point:** At Time 6

Advantages:

- 2-PL ensures serializability or no loops

Disadvantages:

- It does not ensure recoverability as it suffers from Cascading Rollback
- It may suffer from deadlock

Timestamp-based Protocols

- There are mainly two Timestamp Ordering Algorithms in DBMS. They are:
  - Basic Timestamp Ordering
  - Thomas Write rule

(1) **Basic Timestamp Ordering**

The **Timestamp Ordering Protocol** arranges the transactions based on their respective t**imestamps**. A timestamp is a **numeric value assigned** to a transaction when it arrives in a schedule.

Rule No. 01 is used when any transaction wants to perform a **Read(A)** operation

- If WTS(A) > TS (Ti), then Ti Rollback
- Else (otherwise) execute R(A) operation and **SET** RTS (A) = MAX {RTS(A), TS(Ti)}

Timestamp-based Protocols

Rules No.2 rules are used when a transaction needs to perform **WRITE (A)**

- If RTS(A) > TS (Ti), then Ti Rollback
- If WTS(A) > TS (Ti), then Ti Rollback
- Else (otherwise) execute W(A) operation and **SET** WTS (A) = TS(Ti)

Where **“A”** is some data

Example of Timestamp Ordering Protocol

Let's Explain with an Example; look at the following table

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

Solution:

Draw the following table

–	A	B	C
RTS	0	0	0
WTS	0	0	0

In the above table, A, B, and C are data values. And Read and Write timestamp values are given "0". As in the example table, time0 to time7 are given. Let's discuss it one by one.

At time 1, transaction 1 wants to perform a read operation on data "A." then, according to **Rule No 01,**

- WTS(A) > TS(T1) = 0>100 // condition false
- Go to else part and **SET** RTS (A) = MAX {RTS(A), TS(T1)} So,
- RTS (A) = MAX{0,100}= 100.
- So, finally RTS(A) is updated with 100

e updated table will appear as follows,

—	A	B	C
RTS	100	0	0
WTS	0	0	0

At time 2, transaction 2 wants to perform a read operation on data **"B."** then, according to **Rule No 01,**

- WTS(B) > TS(T2) = 0>200 // condition false
- Go to else part and **SET** RTS (B) = MAX {RTS(B), TS(T2)} So,
- RTS (B) = MAX{0,200} = 200.
- So, finally RTS(B) is updated with 200

The updated table will appear as follows,

—	A	B	C
RTS	100	200	0
WTS	0	0	0

At time 3, transaction 1 wants to perform a write operation on data **"C."** then, according to **Rule No 02,**

- RTS(C) > TS(T1) = 0>100 // condition false
- Go to second condition, WTS(C) > TS(T1) = 0>100 // again condition false
- Go to the else part and **SET** WTS (C) = TS(T1) So,
- WTS (C) = TS(T1) = 100.
- So, finally WTS(C) is updated with 100

The updated table will appear as follows,

—	A	B	C
RTS	100	200	0
WTS	0	0	100

At time 4, transaction 3 wants to perform a read operation on data **"B."** then, according to **Rule No 01,**

- WTS(B) > TS(T3) = 0>300 // condition false
- Go to else part and **SET** RTS (B) = MAX {RTS(B), TS(T3)} So,
- RTS (B) = MAX{200,300} = 300.
- So, finally, RTS(B) replaced 200 and updated it with 300.

The updated table will appear as follows,

—	A	B	C
RTS	100	200	0
WTS	0	0	100

At time 5, the transaction T1 wants to perform a read operation on **data "C"** Then according to **Rule No. 01,**

- WTS(C) > TS(T1) = 100>100 // condition false
- Go to else part and **SET** RTS (C) = MAX {RTS(C), TS(T1)} So,
- RTS (A) = MAX{0,100}= 100.
- So, finally RTS(C) is updated with 100

he updated table will appear as follows,

—	A	B	C
RTS	100	300	100
WTS	0	0	100

At time 6, transaction 2 wants to perform a write operation on data **"B."** then, according to **Rule No 02,**

- RTS(B) > TS(T2) = 300>200 // condition True

According to Rule 2, if the condition is true, then Rollback T2.

When T2 rolls, it never resumes. It will restart with a new timestamp value. Keep in mind that T2 restarts after completion of all running transactions, so in this example, T2 will restart after completion of T3.

It happens due to conflict where the older transaction (T2) wants to perform a write operation on data "B," but the younger transaction (T3) has already Read the same data "B"

The table will remain the same

—	A	B	C
RTS	100	300	100
WTS	0	0	100

At time 7, transaction 3 wants to perform a write operation on data **"A"** Then according to **Rule No 02,**

- RTS(A) > TS(T3) = 100>300 // condition false
- Go to second condition, WTS(A) > TS(T3) = 100>300 // again condition false
- Go to the else part and **SET** WTS (A) = TS(T3) So,
- WTS (A) = 300.
- So, finally WTS(A) is updated with 300

An updated table will appear as follows,

—	A	B	C
RTS	100	300	100
WTS	300	0	100

Timestamp Ordering Protocol

Advantage

- The Timestamp protocol ensures serializability and Deadlock removal because a transaction with a smaller timestamp (TS) executes before a transaction with a higher TS.

Disadvantage

- The schedule may not be recoverable.
- The schedule may not be cascading-free.

Data Backup

Loss of Volatile Storage:

Volatile storage like RAM stores all the active logs, disk buffers, and related data.

In addition, it stores all the transactions that are being currently executed.

What happens if such a volatile storage crashes

Following techniques may be adopted in case of loss of volatile storage −

.We can have **checkpoints** at multiple stages so as to save the contents of the database periodically.

.A state of active database in the volatile memory can be periodically **dumped** onto a stable storage, which may also contain logs and active transactions and buffer blocks.

.<dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

Recovery

- When the system recovers from a failure, it can restore the latest dump.

- It can maintain a **redo-list** and an **undo-list** as checkpoints.

- It can recover the system by consulting undo-redo lists to restore the state of all transactions up to the last checkpoint.

Database Backup & Recovery from Catastrophic Failure

A disastrous failure is one where a stable, secondary storage device gets corrupt.

With the storage device, all the valuable data that is stored inside is lost.

We have two different strategies to recover data from such a catastrophic failure −

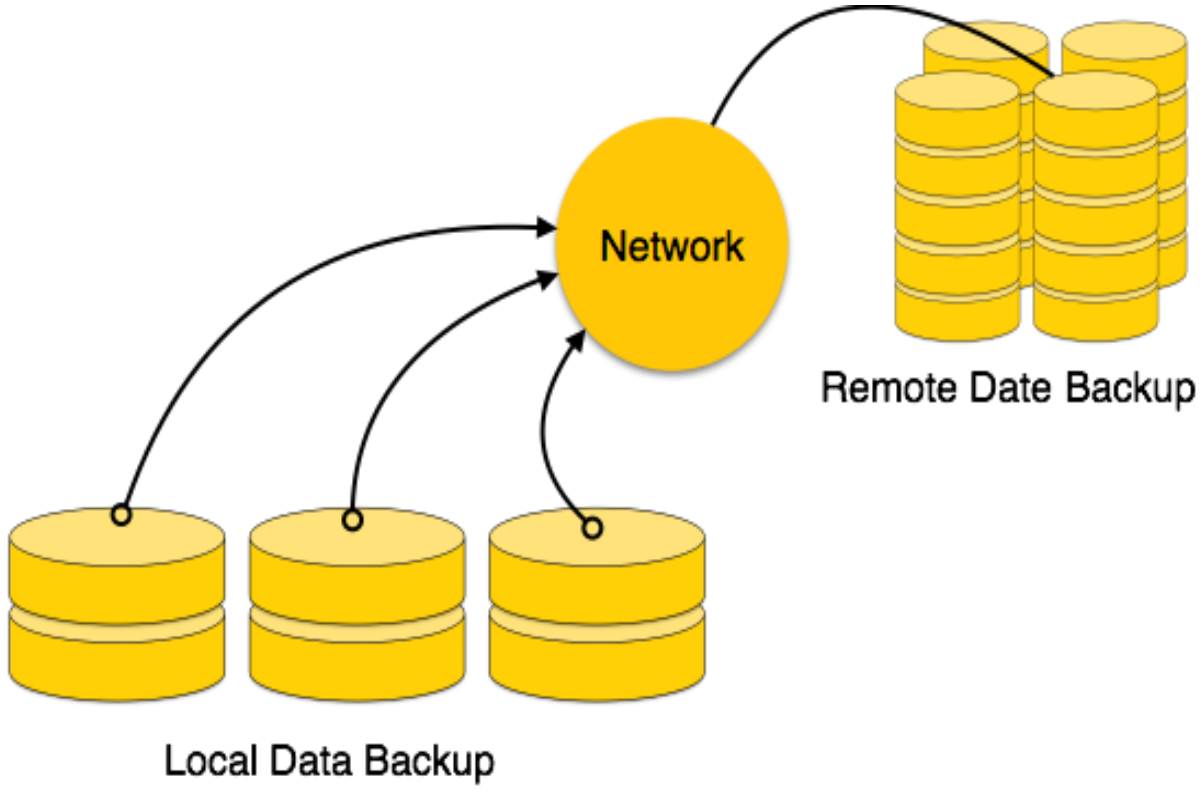
.Remote backup: Here a backup copy of the database is stored at a remote location from where it can be restored in case of a catastrophe.

.Alternatively, database backups can be taken on magnetic tapes and stored at a safer place. This backup can later be transferred onto a freshly installed database to bring it to the point of backup.

Remote Backup

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed.

Remote backup can be offline or real-time or online. In case it is offline, it is maintained manually.



- Online backup systems are more real-time and lifesavers for database administrators and investors.

- An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places.

- One of them is directly connected to the system and the other one is kept at a remote place as backup.

- As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage.

- Sometimes this is so instant that the users can’t even realize a failure.

Data Recovery

1. Crash Recovery:

DBMS is a highly complex system with hundreds of transactions being executed every second.

The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software.

If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification:

1. Transaction failure
2. System Crash
3. Disk Failure

Transaction failure

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

Reasons for a transaction failure could be −

.**Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.

.**System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems − external to the system − that may cause the system to stop sharply and cause the system to crash.

For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.

Transactions are made of various operations, which are atomic in nature.

But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

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

- It should check the states of all the transactions, which were being executed.

- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.

- It should check whether the transaction can be completed now or it needs to be rolled back.

- No transactions would be allowed to leave the DBMS in an inconsistent state.

Log-based Recovery

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 modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

The log file is kept on a stable storage media.

When a transaction enters the system and starts execution, it writes a log about it.

<T_n, Start>

When the transaction modifies an item X, it write logs as follows −

<T_n, X, V_1, V_2>

It reads T_n has changed the value of X, from V_1 to V_2.

When the transaction finishes, it logs −
<T_n, commit>

Recovery with Concurrent Transactions

When more than one transaction 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, and then start recovering.

To ease this situation, most modern DBMS use the concept of 'checkpoints'

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system.

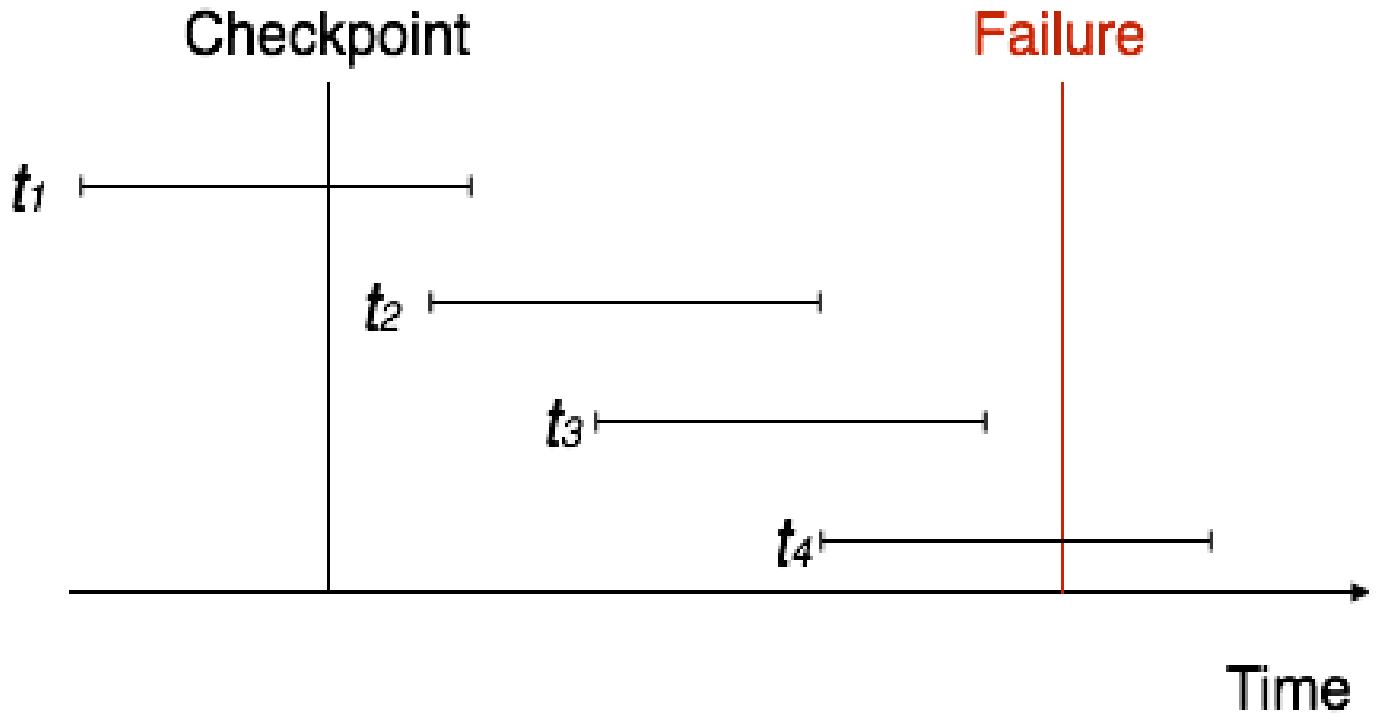
As time passes, the log file may grow too big to be handled at all.

Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.

Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and 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 and a redo-list.
- If the recovery system sees a log with

  $<T_n$, Start> and $<T_n$, Commit> or just

  $<T_n$, Commit>,

  it puts the transaction in the redo-list.
- If the recovery system sees a log with

  $<T_n$, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed.

All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

Consider the following log sequence of two transactions on a bank account: With an initial balance of 12000, transfer 2000 to a mortgage payment and then apply a 5% interest.

1.T1 start
2.T1 B old=12000 new=10000
3.T1 M old=0 new=2000
4.T1 commit
5.T2 start
6.T2 B old=10000 new=10500
7.T2 commit

If the database crashes at point 7, everything until point 4 (including) is already on disk. That is the guarantee that commit gives.

During recovery basically the state after point 4 is restored.

Immediate Update:

- It is a technique for the maintenance of the transaction log files of the DBMS. It is also called UNDO/REDO technique.
- It is used for the recovery of transaction failures that occur due to power, memory, or OS failures.
- Whenever any transaction is executed, the updates are made directly to the database and the log file is also maintained which contains both old and new values.
- Once the commit is done, all the changes get stored permanently in the database, and records in the log file are thus discarded.
- Once rollback is done the old values get restored in the database and all the changes made to the database are also discarded. This is called the “Un-doing” process.

Deferred Update

- It is a technique for the maintenance of the transaction log files of the DBMS. It is also called NO-UNDO/REDO technique.
- It is used for the recovery of transaction failures that occur due to power, memory, or OS failures.
- Whenever any transaction is executed, the updates are not made immediately to the database.
- They are first recorded on the log file and then those changes are applied once the commit is done. This is called the “Re-doing” process.
- Once the rollback is done none of the changes are applied to the database and the changes in the log file are also discarded.
- If the commit is done before crashing the system, then after restarting the system the changes that have been recorded in the log file are thus applied to the database.

Difference between Deferred update and Immediate update:

Deferred Update	Immediate Update
In a deferred update, the changes are not applied immediately to the database.	In an immediate update, the changes are applied directly to the database.
The log file contains all the changes that are to be applied to the database.	The log file contains both old as well as new values.
In this method once rollback is done all the records of log file are discarded and no changes are applied to the database.	In this method once rollback is done the old values are restored to the database using the records of the log file.
Concepts of buffering and caching are used in deferred update method.	Concept of shadow paging is used in immediate update method.
The major disadvantage of this method is that it requires a lot of time for recovery in case of system failure.	The major disadvantage of this method is that there are frequent I/O operations while the transaction is active.
In this method of recovery, firstly the changes carried out by a transaction on the data are done in the log file and then applied to the database on commit. Here, the maintained record gets discarded **on rollback** and thus, not applied to the database.	In this method of recovery, the database gets directly updated after the changes made by the transaction and the log file keeps the old and new values. In the case of **rollback**, these records are used to restore old values.

Shadow Paging:

Shadow paging is one of the techniques that is used to recover from failure. We all know that recovery means getting back the information, which is lost. It helps to maintain database consistency in case of failure.

Concept of shadow paging:

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.