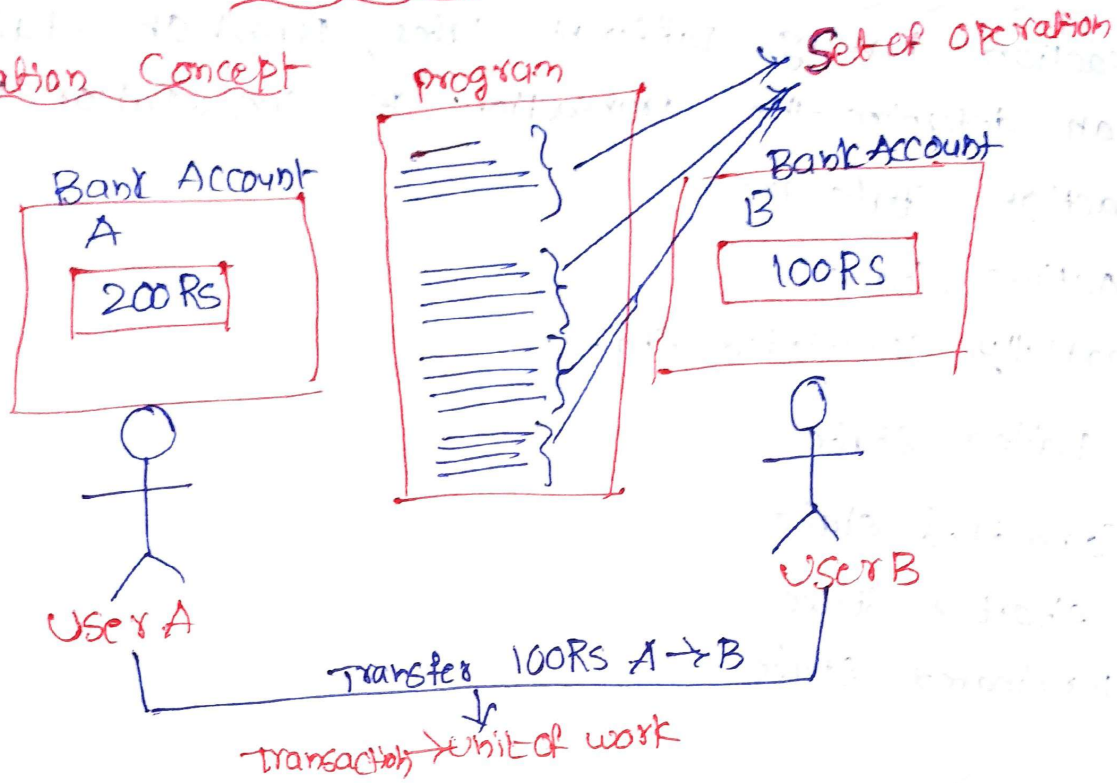


UNIT-IV

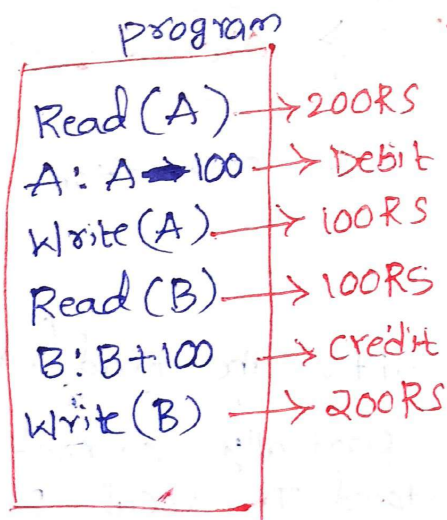
①

Transaction Concept



Database Transaction:

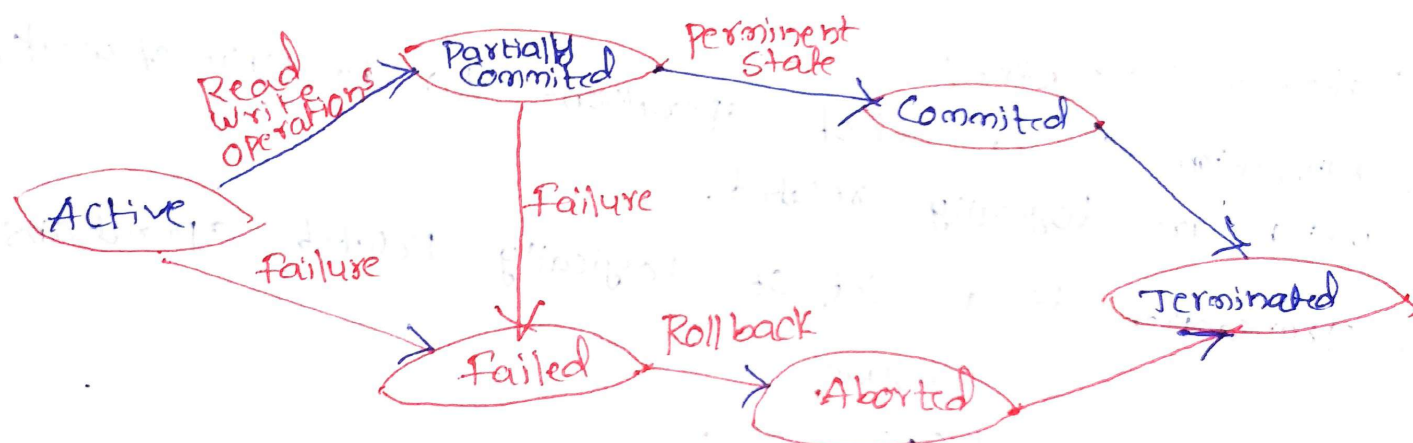
- Transaction is a set of operations to do unit of work, which are logically related.
- Transaction is a set of logically related operations



- Here Read and write are the operations
- Logic — Debit and credit

Transaction State: Transaction is series of operations. ☺
Transaction having different states, Based on state we can determine the transaction is successful (or) not.
Transaction states are,

1. Active state
2. Partially committed state
3. Failed state
4. Committed state
5. Aborted state
6. Terminated state



Active state → In this state transaction is Read to Read/write operation

Partially Committed state → After the Read/write operations transaction is coming to partially committed state. In this state transaction is stored in local buffer

Failed state → When ever the transaction is failed in Active state, it will come to failed state.

When ever the system failure occurs in partially committed state, the transaction also come to the failed state.

③
4. Committed State :- A Transaction is after
a) Partially Committed state It will go to Permited state
and get committed state. ~~Transaction~~

→ Transaction may satisfy all the "ACID" Properties
Then only it stored in Database.

→

5. Aborted State :-

→ When ever the Transaction failed, Roll back is
Clear the Data in the Local Buffer and it will
Come to Aborted State

→ In Aborted state the Transaction Values Keep on change.

→ After that It will send the Transaction to
Terminated ~~set~~ state with out changing the Values.

→ Consistent data should be send from Aborted state
to Terminated state.

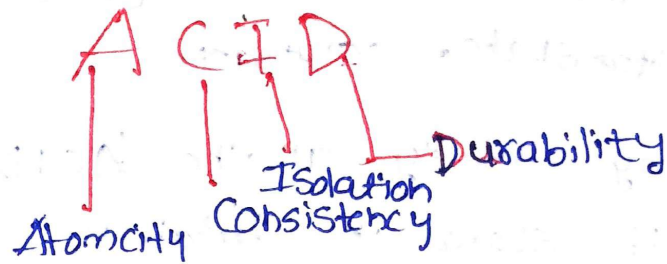
6. Terminated State :-

→ After completion of committed state (or) Aborted
State, the transaction is enter to the Terminated state

→ In the terminated state, The Data must be Consistent.

ACID PROPERTIES

④



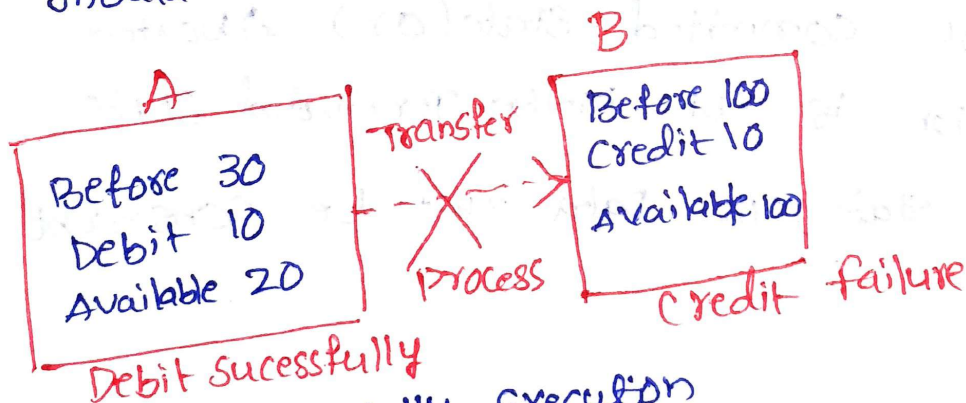
ACID Properties

There are four ACID properties of Transaction

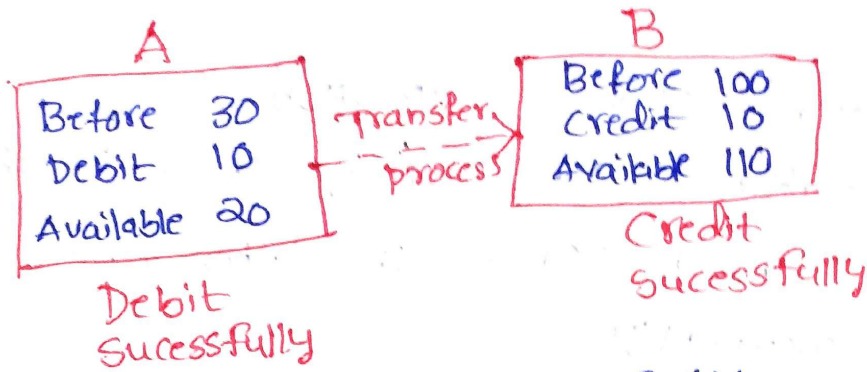
1. Atomicity
2. Consistency
3. Isolation
4. Durability

1. Atomicity:-

→ The term atomicity defines that the data remains Atomic
→ It means if any operation is performed on the data, either it should be performed (or) executed completely (or) should not be executed at all



- Partially Execution
- No Atomicity,
-

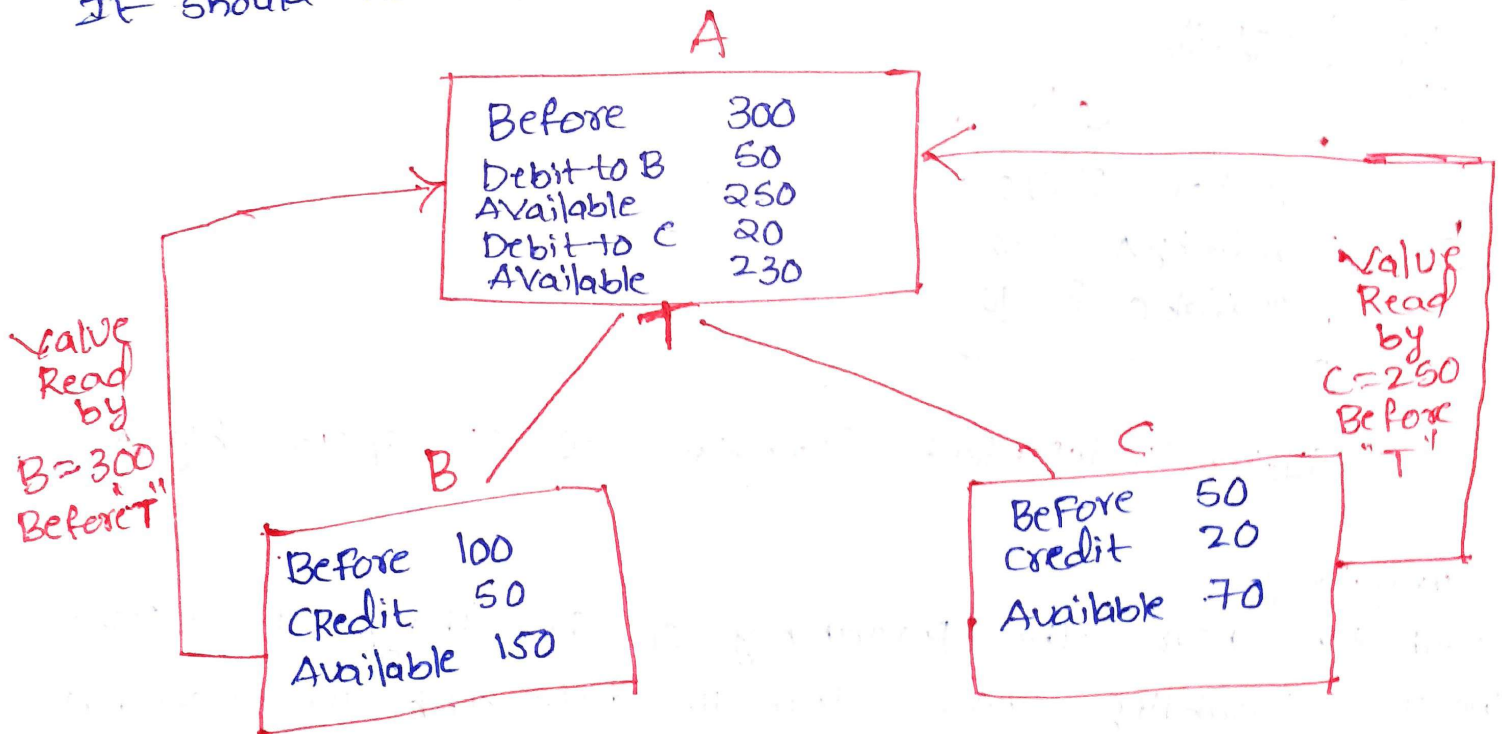


Execution Done Successfully

Atomicity Done

② Consistency :-

- The word Consistency Means that the value should remain preserved always.
- In DBMS, the Integrity of data should be maintained, which means if a change in the Database is made It should remain preserved always.

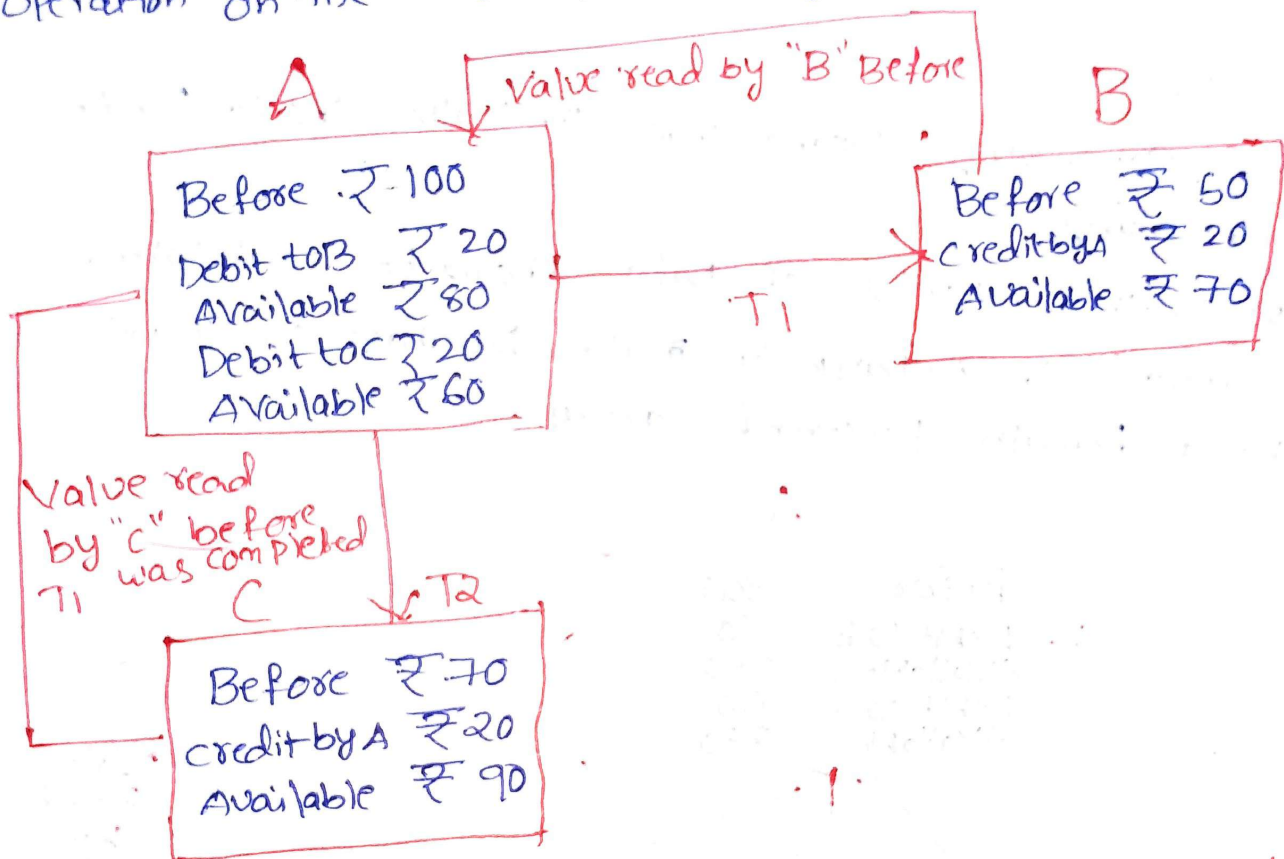


③ Isolation :-

→ The term "Isolation" Means Separation.

→ In DBMS, Isolation is the property of a Database where no Data should effect the other one and may occur Concurrently

→ The Operation on one Database should begin when the Operation on the first Database gets complet



Isolation-Independent Execution of T1 & T2 by A

④ Durability :-

Durability ensures the permanency of some thing, In DBMS

The term Durability ensures that the data after the successful execution of operation becomes permanent in Database. The durability of Data should be so perfect that even if the system fails (or) lead to a crash.

→ Recovery manager is Responsible for durability of Database

→ COMMIT command must be used for every time we make changes.

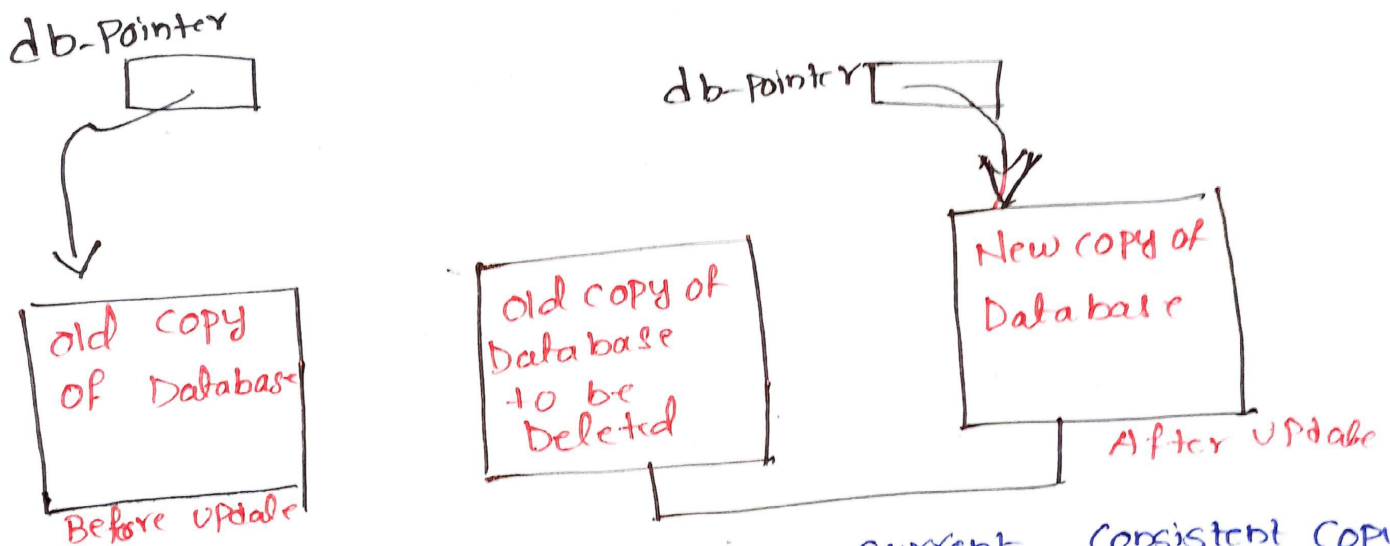
Implementation of Atomicity and Durability

(2)

→ The Recovery Management Component of database system implements the support for atomicity and durability

→ eg:- The Shadow-Database Scheme. It all support updates are made on shadow copy of database.

→ db-Pointer is made to point to the updated shadow copy after the transaction reaches partial commit and all updated pages have been flushed to disk



→ db-Pointer always points to the current consistent copy of the database

→ In case transaction fails, old consistent copy pointed to by db-Pointer can be used, and shadow copy can be deleted

→ The Shadow-Database Scheme

Assumes that only one transaction is active at a time

Assumes disks do not fail

Does not handle concurrent transactions.

Concurrency Control

Concurrency!

Allowing multiple transactions to run concurrently on the same database.

Concurrent Execution: In multiuser system, multiple users can access and use the same database at one time.

Problem with Concurrency Execution!

In a database transaction, the two main operations are READ and WRITE

→ There is need to manage these two operations in concurrent execution of the transactions and if these operations are not performed in an individual manner, and data may become inconsistent.

→ The following problems occur with the concurrent execution of the operation.

1. Last update (W-W conflict)
2. Dirty Read problem (W-R conflict)
3. UnRepeatable Read problem (R-R conflict)

Concurrency Control!

Concurrency control is a crucial DBMS component. It manages simultaneous operations without their conflicting with each other.

→ The primary aim is maintaining Consistency, Integrity and Isolation when multiple Users (Or) Applications Access the Database Simultaneously. (6)

Advantages:

- Writing time will be Decreased.
- Response time will be Decreases
- Resource ~~utilite~~ utilization will Increases
- System performance & Efficiency is increased.

Serializability and Recoverability:

Schedule:

A Sequence of Operations by a set of Concurrent Transactions that Preserves the order of Operations, in each of the Individual Transactions

Serial Schedule:

A schedule where the operation of each transaction are Executed Continuously without any Interrelated operations from other transaction.

Non-Serial Schedule:

A schedule the operation from a set of Concurrent Transactions are Inter Related.

Serializable schedule:

The objective of Serializability is to find Non Serial schedules that allow Transactions to execute Concurrently without Interfering one another, and there produce database state that could be produced by Serial Execution that means a Non-Serial Schedule is Serializable and correct if it produces same Result.

Time	T ₁	T ₂	T ₁	T ₂	T ₁	T ₂
t ₁	begin Transaction		begin_Trans		begin_tr	
t ₂	Read(bal x)		read(bal x)		read(bal x)	
t ₃	Write(bal x)		write(bal x)		write(bal x)	
t ₄	begin		begin_tr		read(bal y)	
t ₅	Read bal(x)		read(bal x)		write(bal y)	
t ₆	Write bal(x)		Read(bal x)		Commit	
t ₇	Read(bal y)		write(bal x)		begin_tr	
t ₈	Write(bal y)		Write(bal x)		read(bal x)	
t ₉	Commit		Commit		write(bal x)	
t ₁₀	read bal(y)		read(bal y)		read(bal y)	
t ₁₁	Write bal(y)		write(bal y)		Write(bal y)	
t ₁₂	Commit		Commit		Commit	
Non-Serial Schedule S1			Non Serial Schedule S2		Serial Schedule S3	

Lock Based Protocol (or) Locking Technique (11)

for Concurrency Control

A Lock is nothing but a Mechanism that tells the DBMS whether a particular Data Item is being used by transaction for Read/Write purpose

Type of operations -

1. READ operation
2. WRITE operation.

→ Any Transaction cannot READ (or) WRITE data until it acquires an appropriate Lock on it.

→ If the Transaction is reading the content of a sharable item then any number of other processes can be allowed to Read the content of same Data item.

→ But if any Transaction is writing in to sharable data item then no other Transaction will be allowed to Read (or) Write that same Data Item.

Types of Locks :

1. Shared Lock
2. Exclusive Lock.

1. Shared Lock: It is also known as Read-only Lock

→ A Transaction may acquire Shared Lock on a data item in order to Read it.

→ The lock is shared in the sense that any other Transaction can acquire the shared lock on that same Data item for Reading purpose.

2. EXCLUSIVE LOCK:

(12)

A Transaction may acquire EXCLUSIVE LOCK on a Data item in order to Both READ and WRITE in to it

→ The Lock is Exclusive in the sense that NO other Transaction can acquire any kind of Lock on that same Data Item.

Example for Shared Lock:

LOCK - S(A); (shared lock)

Read - A;

Temp = A * 0.1;

unlock(A)

LOCK - x(C);

Read - C;

C = C + temp;

Write(C)

unlock - (C)

Example for Exclusive Lock:

LOCK - x(A); (Exclusive Lock)

Read - A;

A = A - 100;

Write - A;

unlock(A); (Unlocking, A after modification is done)

LOCK - x(B);

Read - B;

B = B + 100;

Write - B;

unlock(B);

Categories of Two Phase Locking:

There are three ways categories of 2 phase Locking.

1. Strict Two phase Locking protocol
2. Rigorous Two phase Locking protocol
3. Conservative Two phase Locking protocol

1. Strict Two phase Locking protocol:

- This Requires that in addition to the Lock being 2-phase
- In strict 2 phase Locking protocol if one Transaction Rollback then other transaction should also have to Roll back.
- The transaction are dependent on each other. This is called cascading schedule.

Ex1:

T ₁	T ₂
L-x(A)	
W(A)	
R(A)	
Commit	
U(A)	
	L-S(A)
	R(A)
	U(*)

Ex2:

T ₁	T ₂
L-x(A)	
(LOCK)	L-x(B)(LOCK)
	→ Dead lock
L-x(B)	
(Request)	L-x(A)(Request)

2. Rigorous Two phase Locking protocol:

- This Request that in addition to the lock being 2-phase all locks (Exclusive and shared) held by transaction be released until after the transaction commit.
- The transaction cannot Release either of the locks i.e. neither shared lock nor Exclusive lock.

Ex:

T ₁	T ₂
L-X(A)	
L-S(B)	L-X(C)
⋮	L-S(D)
Commit	Commit
U(A)	U(C)
U(B)	U(D)

(14)

3. Conservative Two Phase Locking protocol:

- The Transaction must lock all the data items it requires in the transaction before the transaction begins.
- If any of the data item is not available for locking before execution of the lock, then no data items are locked.
- It is a deadlock free.

Time stamp ordering protocol :

(15)

Time stamp is a tag that can be attached to any transaction (or) any data item, which denotes a specific time on which the transaction (or) data item activated.

- A Time stamp is a unique Identifier that is created by DBMS when Transaction enters in to the system.
- It is used to order the transactions based on their timestamps.
- The protocol uses system clock (or) logical counter to execute higher priority transaction first.
- The Time stamp ordering protocol also maintain the Time stamp of last "READ" and "WRITE" operation on a data.

Example :

Transaction	T ₁	T ₂	T ₃
Time Stamp	10	20	30
priority (high to low)	older		Younger

TS(T_i) - Indicates Timestamp of Transaction T_i
 EX - T₁ : 10 , T₂ : 20 , T₃ : 30

R-TS(A) :

Last (latest) Transaction no, which perform Read Successfully

Transaction : T₁ T₂ T₃
 Timestamp : 10 (old) - 30 (Young)
 RCA) RCA) RCA)

R-TS(A) = "30"

R-TS(A)

W-TS(A) ÷ Last (latest) Transaction no, which perform write successfully.

EX:-

Transaction :	T ₁	T ₂	T ₃
Time stamp :	10 (old)	20	30 (young)

W(A)

W-TS(A)

W-TS(A) = 20 →

Read Rule

X	T ₁	T ₂
	10 (old)	20 (y)
	R(A)	W(A)
	A = 10	A = 10
		A = A + 10
		A = 20

T ₁	T ₂
10 (old)	20 (young)
W(A)	
Commit	R(A)

Write Rule

T ₁	T ₂
10 (old)	20 (young)
R(A) / W(A)	
Commit	W(A)

Validation Based Protocol

(17)

- Validation based protocol in DBMS also known as optimistic concurrency control Technique.
- It is a Method to avoid concurrency in Transactions.
In this protocol the local copies of Transaction Data are updated rather than the database itself, which result in less interference while execution of Transaction.
- In this we have three Phases
 1. Read phase
 2. Validation phase
 3. Write phase

1. Read Phase:

- In the Read phase, the Data values from the Database can be Read by Transaction but the WRITE operation (or) updates are only applied to local Data copies, not actual Database.

2. Validation Phase:

- In Validation phase, the data is checked to ensure that there is no violation of Serializability while applying the Transaction updates to the Database.

3. WRITE Phase:

- In the Write phase, the updates are applied to the Database if the Validation is successful. else the updates are not applied and the Transaction is Rolled back.

To clear all validation test by T_i , then T_j must satisfy one of the following conditions.

(i) $\text{Finish}(T_i) < \text{Start}(T_j)$:

It Means T_i is older transaction & it gets finished before T_j starts. (No overlap)

S	V	F
5	10	15

S	V	F
19	29	39

(ii) $\text{Finish}(T_i) < \text{Validate}(T_j)$:

Ensures actual write by T_i & T_j will not overlap.

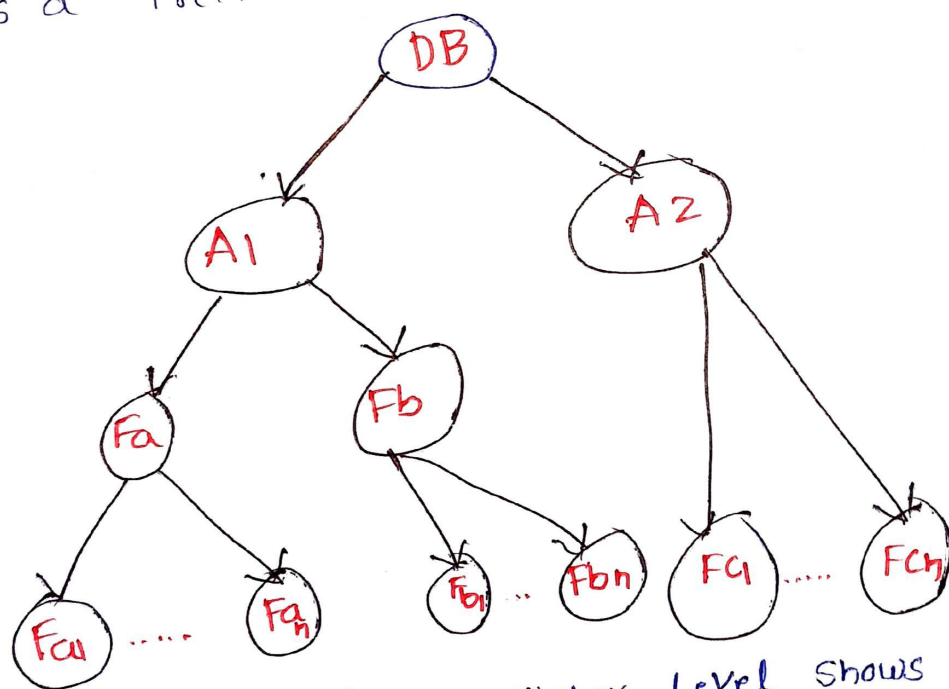
S	V	F
5	10	15

S	V	F
19	29	39

(iii) $\text{Validate}(T_i) < \text{Validate}(T_j)$:
Ensures that T_i has completed read phase before T_j completes read phase.

Multiple Granularity:

- It can be defined as hierarchically Breaking up the Database into blocks which can be locked.
- The Multiple Granularity protocol Enhances concurrency and Reduces lock overhead.
- It Maintains the Track of what to Lock and How to Lock.
- It Makes easy to Decide either to Lock a data item (or) to unlock a Data item.
- This Type of hierarchy can be Graphically represented as a Tree.



- The first level (or) Higher Level shows the entire Database
- The second Level represents a Node of type area.
- The Area consist of children Nodes which are known as files. No file can be present in more than one Area.
- Finally each file contains child Nodes known as records. The file has exactly those Records that are its child node. No records represent in more than one file.

Hence the level of the tree starting from the top level are as follows.

— Database

— Area

— File

— Record.

There are three additional Lock Nodes with Multiple Granularity.

Insertion-Shared (IS) : It contains Explicit Locking at a lower level of the tree but only with shared locks.

Insertion-Exclusive (IX) : It contains Explicit Locking at a lower level with exclusive (or) shared lock.

Shared & Intention-Exclusive (SIX) :

In this lock the node is locked in shared mode and some node is locked in exclusive mode by the same transaction.

Recovery and Atomicity

(21)

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

— The transactions are made of various operations, which are atomic in nature.

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

→ When a DBMS Recover 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 transaction in this case.

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

→ No transaction would be allowed to leave the DBMS in an inconsistent state.

→ There are two types of techniques which can help a DBMS Recovering as well as maintaining the Atomicity of transaction.

→ ① Maintaining the logs of each transaction. and writing them onto some stable storage Before Actually modifying the database

(2) Maintaining shadow paging, where the changes are Done on the volatile Memory, and later the Actual Database is updated.

Log Based Recovery

- Log is a Sequence of Records, which maintains the Records of Actions performed by transaction.
- 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 Data base.

- Any update log record represented as

$\langle T_i, X_j, V_1, V_2 \rangle$ has these fields.

Transaction Identifier: Unique Identifier of the transaction that performed the write operation

Data Item: Unique Identifier of the data item written.

Old value: Value of Data Item prior to write

New value: Value of Data item after write operation.

Let's Assume there is a transaction to modify the city of student. The following logs are written for the transaction.

→ When the transaction is initiated, then it write 'start' log. $\langle T_n, \text{start} \rangle$

→ When the transaction modifies the city from "Noida" to 'Banglore'. Then another log is written to the file.

$\langle T_n, \text{city}, 'Noida', 'Banglore' \rangle$

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

$\langle T_n, \text{commit} \rangle$

Approaches to Modify Database

1. Deferred Database Modification:

- The Deferred Modification Technique occurs if the transaction does not modify the database until it has committed.
- In this method all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate Database Modification:

- The Immediate modification Technique occurs if database modification occurs while the transaction is still active.
- In this technique the database is modified immediately after every operation. It follows an actual modification.

→ When the system is crashed, then the system consults the log to find which transaction need to be undone and which need to be redone.

→ If a log contains the records $\langle T_i, \text{start} \rangle$ 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 $\langle T_i, \text{start} \rangle$ but does not contain the record either $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$, then transaction T_i needs to be undone.

Recovery with Concurrent Transaction

→ 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 back-track all logs and then start recovering.

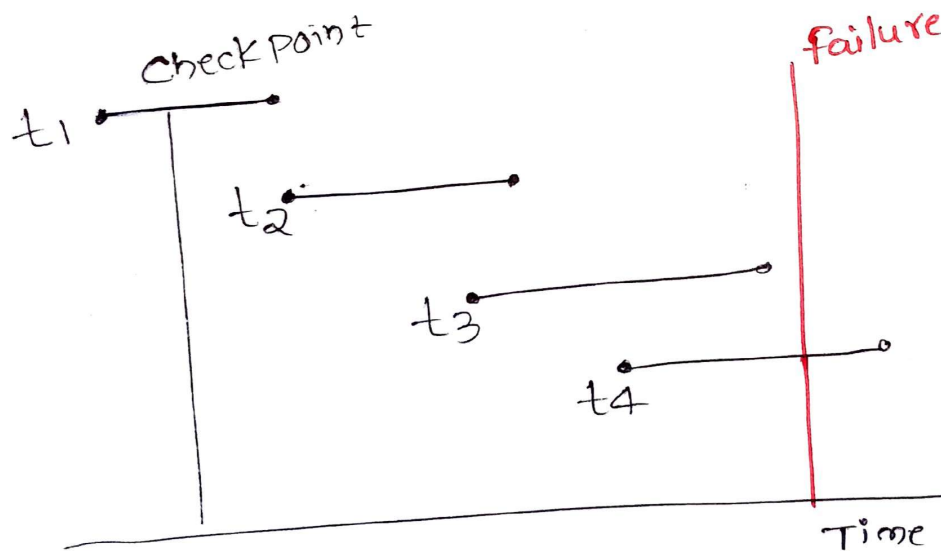
→ To ease this situation, most modern DBMS use the concept of 'check points'.

Check Points

Keeping and Maintaining Logs in Real time and real Environment may fillout 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. Check point declares a Point Before which the DBMS was in consistent state, and all the transactions were committed.

→ When a system with Concurrent Transactions crashes and Recovers it behaves the following Manner.



→ The Recovery system reads the Logs backwards from the End to the last Check point

→ The Recovery system reads log files from the end to start, It reads log files from T_4 to T_1

— Recovery system maintains two lists,
a redo-list and an un-do 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 T_2 and T_3 will have $\langle T_n, \text{start} \rangle$ and $\langle T_n, \text{commit} \rangle$, The " T_1 " Transaction will have only $\langle T_n, \text{commit} \rangle$ in the log file. That's why the transaction is committed after the check point crossed. Hence it puts T_1 , T_2 and T_3 Transaction in to redo list.

→ The Transaction is put into state if the Recovery system sees a log with $\langle T_n, \text{start} \rangle$ but no commit or abort log found. In the un-do list, all the transactions are undone, and their logs are removed.

→ For Eg: Transaction T_4 will have $\langle T_n, \text{start} \rangle$ So T_4 will be put into undo list since the transaction is not yet complete and filed a mid.

Buffer Management :

- Buffer management acts as a bridge between fast, but limited, primary memory (RAM) and slower, but larger, secondary storage such as a Hard Disk.
- The main Goal is to Reduce the Number of disk accesses by placing frequently accessed data in Buffer Pool.
- Efficient Buffer Management is important because disk access can be a Major bottleneck in DBMS Performance.

Buffer Manager

- The Buffer Manager is component of the DBMS Responsible for managing the Buffer Pool.
- It Decides which data should be kept in the Buffer Pool and for how long.
- When the request for data Arrives, the Buffer manager first checks whether the data is already in the Buffer Pool. If it's there, it's a 'hit', and the data is delivered immediately.
- If not, a 'Miss' occurs, and the Buffer manager brings the data from the Disk into the Buffer Pool.
- Additionally the Buffer Manager must Decide which data to Remove when the Buffer Pool becomes Full. This process known as Replacement Policy.

Methods of Buffer Management:

(28)

There are several methods of Buffer management that mentioned below.

1. LRU (Least Recently Used):

The method removes pages of the ones which do not get written to most since randomly in Buffer Pool.

2. MRU (Most Recently Used):

Unlike LRU is not preserved the most recently used to refresh in MRU. while it may run against the grain, caching the most recent set of data can be time saving because you would not need to again access these data in a short span of time.

3. Clock Policy

Therefore this can be called LRU. Try to think of a clock whose the buffer encompasses all the data pages with a clock hand for each. However the handle for this page is set to '1' if it is being viewed. Each time the Paging system starts, the Buffer manager checks these pointers iteratively, and in this process, the '1' is turned into a '0' until it encounters '0' which it then sends on the sharp page elimination process removes

4. LFU (Least Frequent Accessed):

This method frees up space by setting aside data that is accessed the atleast often, the book store could ponder whether to bring back that book Distributes (or) it can be sold.

Failure with loss of Non Volatile storage

So for we Assumed no loss of Non-Volatile storage

- Technique Like check pointing Used to Deal with the Loss of Non-volatile storage.
- Periodically dump the Entire content of Database to Stable Storage.
- No Transaction may be active during the Dump procedure, a procedure like checkpointing must take place
- Output all the Records currently residing in Main memory onto Stable Storage.
- Output all Buffer blocks onto the disk.
- Copy the contents of Database to stable storage
- Output a record <dump> to Log on stable storage
- **TO Recover from Disk failure**
 - Restore database from most Recent dump
 - Consult the log and redo all transactions that committed after the Dump.
- Extended to allow Transactions to be active during the Dump known as Fuzzy Dump (or) online Dump.