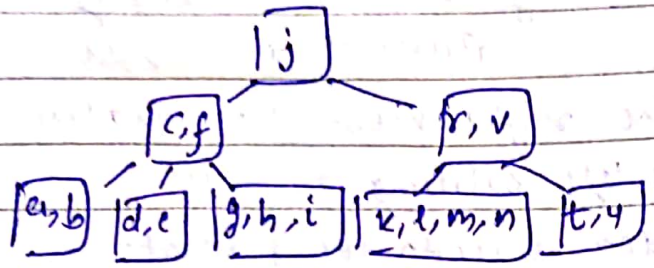
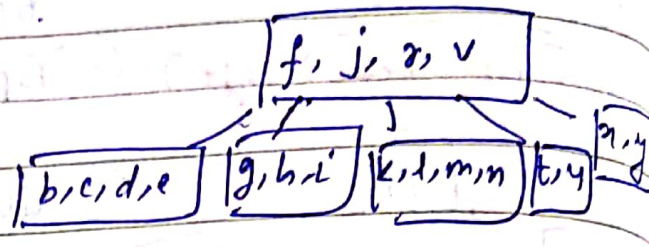
3,12

check underflow instead of deletion
∵ root has ~~has~~ less than n/2

(iv) Delete a from (iii) tree.

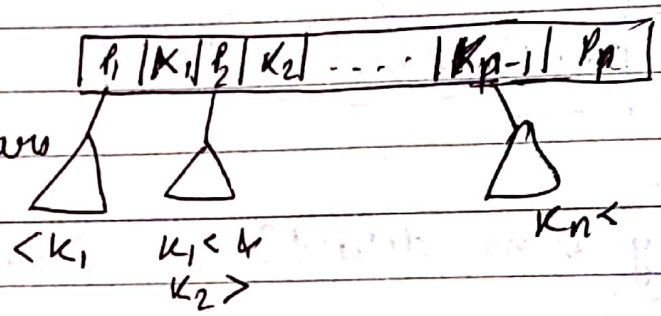




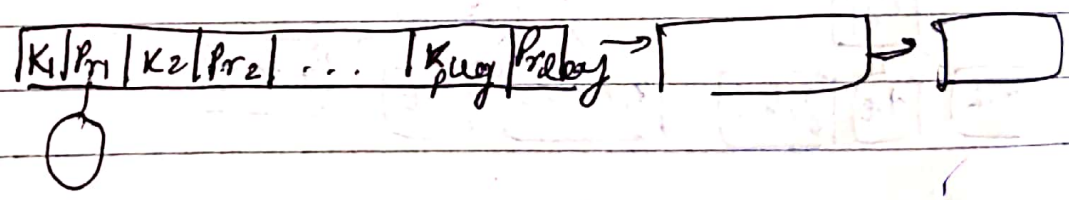
B+ Trees

Internal Nodes

No data pointers



External Node



leaf Nodes are joined using linked list

B+ can store more keys in internal nodes as space for storing data pointer is removed

	~~Nodes~~ keys	Ptrs
Internal Nodes =	p−1	p−1
External Nodes =	P_{leaf}	p−1

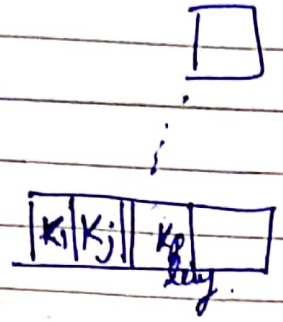
Scanned with CamScanner

No direct insertion in internal node.

First Splitting, Insertion is always at leaf Node

Overflow

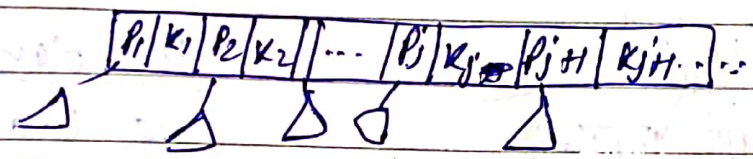
External Node



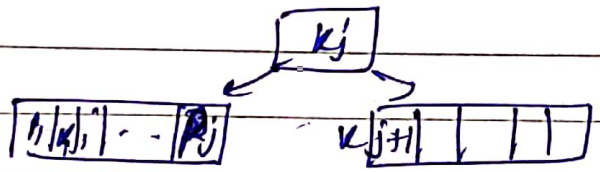
When inserting at leaf Node & overflow occurs then we do not delete it from leaf but replicate it above (B Tree method).
[as B+ has data ptr. for key at leaves]

$$\text{Median } j = \left\lceil \frac{(p+1)}{2} \right\rceil$$

Internal Nodes



$$\text{Median} = \left\lceil \frac{p+1}{2} \right\rceil$$



B Tree (only median change right biased)

Q) 2, 3, 5, 7, 9, 11, 17, 23, 29, 31, 6, 8

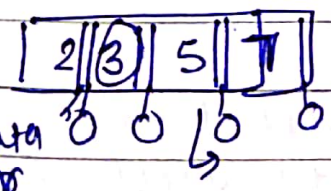
B+ Trees
$p = 4$, $P_{leaf} = 3$

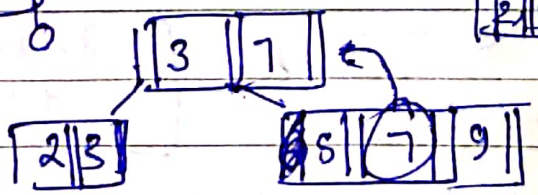
Sol. $p = 4 \therefore p-1$ key $= 3$

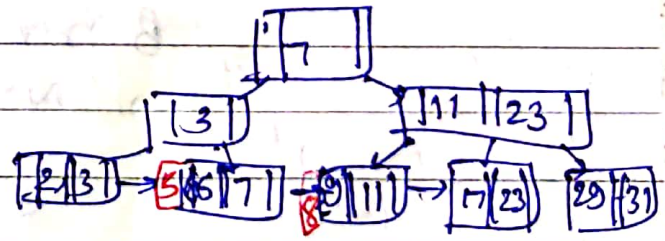
Can be diff.

$p' = 3$ in ~~leaf nodes~~ keys in leaf

Empty nodes is considered leaf Node





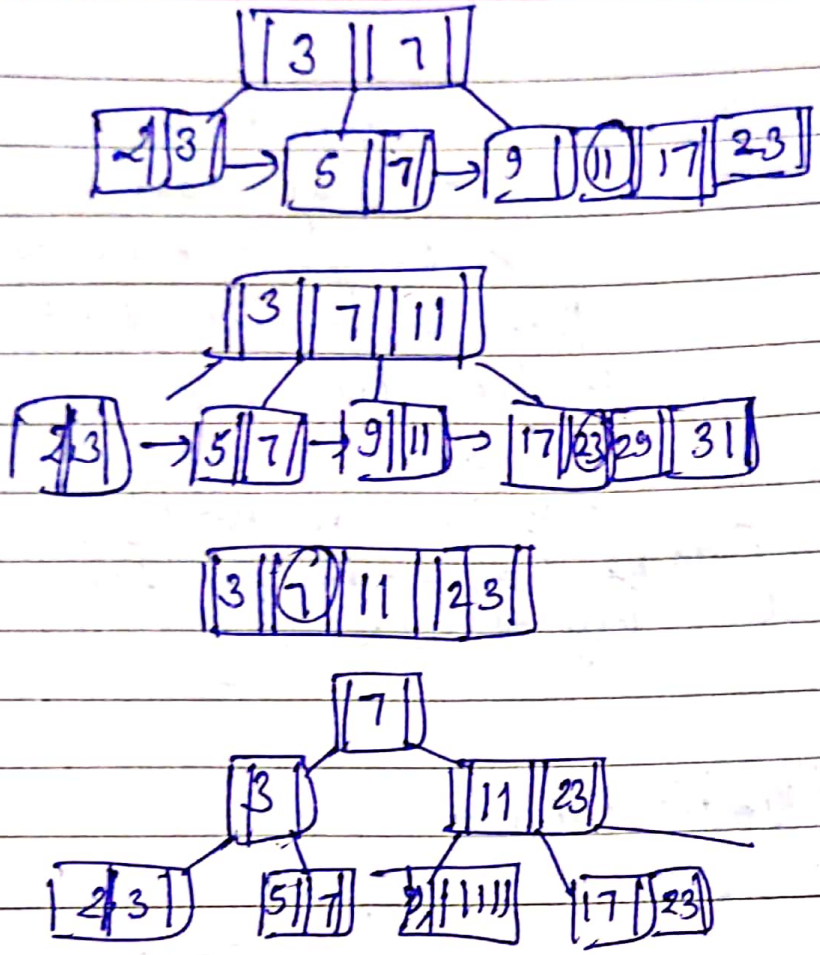


Scanned with CamScanner

Internal Nodes → Guiding purpose

External Nodes → Data Records (Data ptr)

Search will always terminate at leaf nodes



B+ Trees gives facility to go to every nodes

Index sequential

Tree ($O(\log n)$)

Linear Linked list ($O(n)$)

Deletion

At least one (1) key-value ~~for~~ should be in Leaf Node

At least $n/2$ key value ~~to~~ should be in internal nodes

Time Complexity

$\log_q n$

n → B Tree / B+ Trees No. of Nodes in the tree

q → order of the tree

Insertion as well as Deletion

Same as $\log_2 n$

Scanned with CamScanner

24.11.2022
Thursday

Transaction

→ logical unit of DBMS operations
Read, Write

Concurrency reduces time of execution
Waiting time ↓ es.
Response time ↑ es

Interleaving / Intermixing

400 $\xrightarrow{50}$ 500
X → Y

T1
BEGIN
read (X); // putting value in X
X = X - 50; // X = 350
write (X); // updating
read (Y); // Y = 450
Y = Y + 50;
write (Y);
COMMIT
END

BEGIN
↑
Transaction
↓
END

Granularity level of data items
↳ anything
(attribute, tuple, etc)

→ Find memory block of X in secondary block & bring that block in buffer zone of main memory
updated in local copy
but not in DB
(if not present in buffer)

Immediate update
Deferred update

$$\boxed{(X+Y)_{initial} = (X+Y)_{final}}$$

consistency.

The amount of fund transferred before Transaction == Amt. of fund transferred after transaction

Scanned with CamScanner

failure of Transaction in betn;

Due to Disc crash, Network error.

solution : Rollback to byon Transaction

How?

Creating log of all process

→ smallest

Atomicity

Transaction will occurs in its entirity or not at all

Timestamp

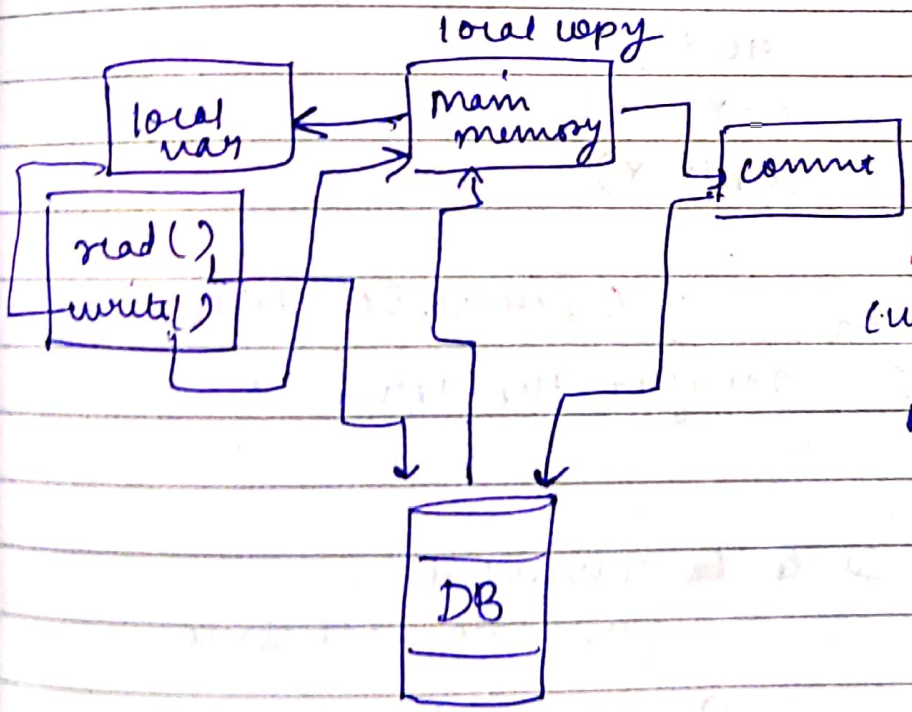
t_0	BEGIN
t_1	X = 350
t_2	X = 320
t_3	X = 300
System fails	

Until found Begin it will roll back.

why?

Else it will leave us to inconsistent state

$(X+Y)_{initial} \neq (X+Y)_{final}$.



Transaction has two states only

ABORT

(undo the effects / Rollback)

COMMIT

(Successful)

send it to original DB

Scanned with CamScanner

→ serial schedule

Scheduled (way of writing)

If write directly

T1	T2
read(X)	
X = X-50	
write(X);	
read(Y);	
Y = Y+50;	
write(Y)	
	read(Y);
	Y = Y+100;
	write(Y);

T1	T2
read(X)	
X = X-50	
write(X)	
	read(Y)
	Y = Y+100
	write(Y)
read(Y)	
Y = Y+50	
write(Y)	

T1	T2
	read(Y)
	Y = Y+100
	write(Y)
read(X)	
X = X-100	
write(X)	

Δ Concurrent Schedule → reduction of time

first one, then second one

then again one

should not end in inconsistent state

$(X+Y)_i \neq (X+Y)_f$

Δ Serial Schedule

One after the other

→ can be converted to Serializable Schedule

⇒ Serializable Schedule (No inconsistency)

Scanned with CamScanner

Problems of Concurrency Scheduling:

Each Transactions has its local variables.

Lost Update Problem:

T_1	T_2	
read(X);		// X = 100
X = X - N;		// X = 100-50=50
	read(X)	// X = 100 (local copy से copy करेगा)
	X = X + M;	// X = 100+60=160
write(X)		// update X in local copy from local variable
	write(X)	// Again same update ∴ Overwriting
read(Y)		
⋮		

Value updated by T_1 has no effect in future steps of Transaction

i.e., updates of T_1 are lost.

Need to avoid such problems.

Occurs in: Concurrent ~~Tra~~ scheduling.

Temporary Update / Dirty Read Occurs:

When ~~next~~ transaction reads uncommited value of a temporary variable & updates it

	T_1	T_2	
100	read(X)		
	X = X - N		
50	write(X)		
		read(X)	50
		X = X + M	
		write(X)	150

Can't be rolled back ∵ X T_1 is lost.

— Failure — → Roll Back

Scanned with CamScanner

→ sum

Incorrect Summary Problem

T_1	T_2
	sum = 0
	read (A)
	sum = sum + A;
	⋮
read (X)	
X = X + N;	
write (X);	
	read (X);
	sum = sum + X
	read (Y);
	sum = sum + Y;
read (Y)	
Y = Y − M	
write (Y)	

Occurs:
Sum operations and updating values for some and for some not

Ex.
Class marks and updating.

Solⁿ: Freeze the updation of any variable until summary operation ends
↳ Locking

Blind Write

(Not of concurrency)

T_2	T_1
⋮	⋮
	write (X)
write (X)	

without reading, just writing

↳ No effect due to this

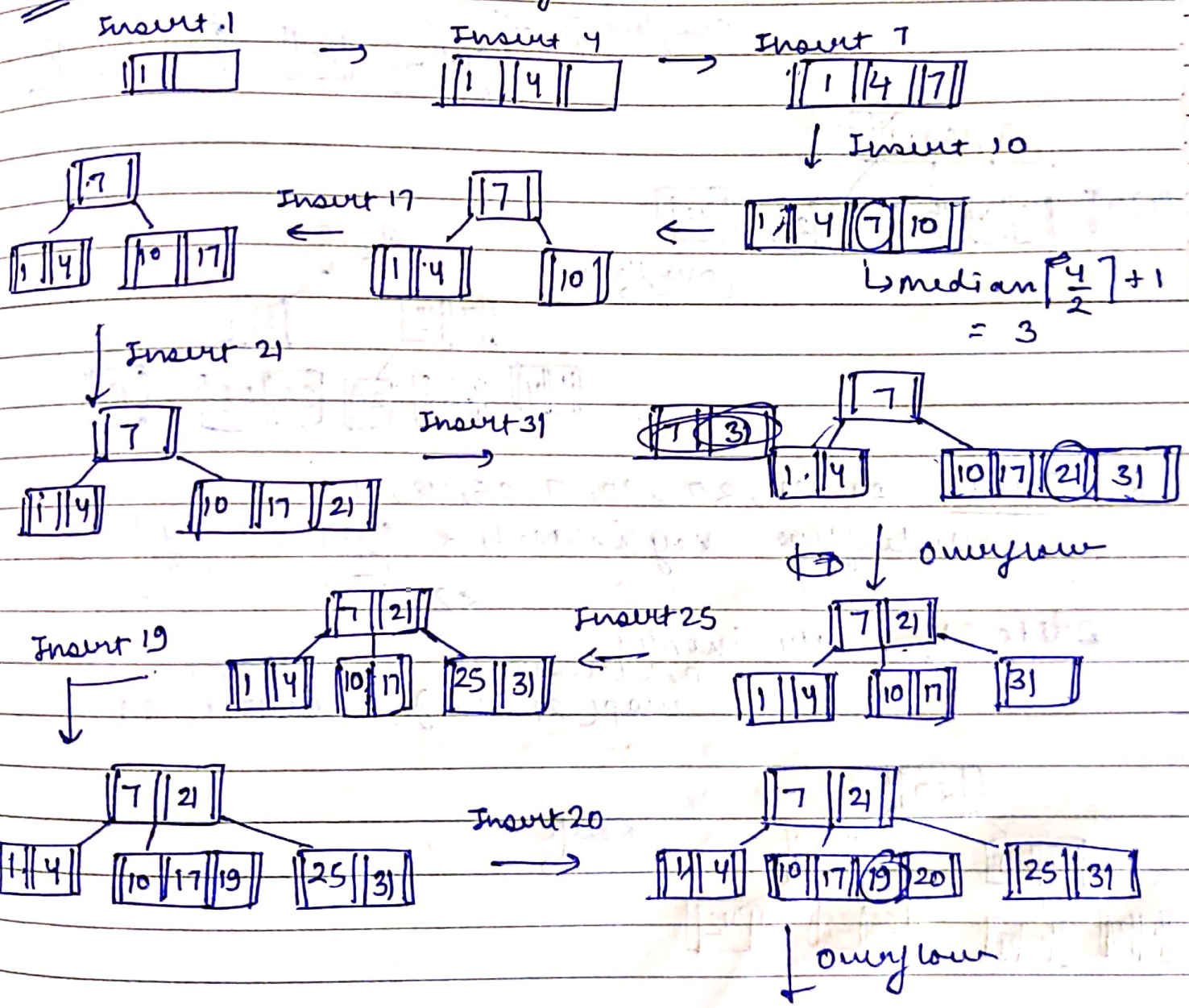
↳ only effect due to this

Scanned with CamScanner

Q.1) Show the construction of B Trees of order 4 with keys 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

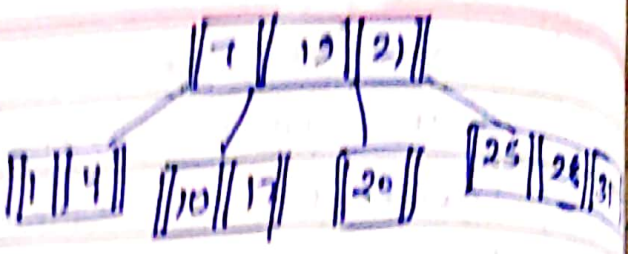
Q.2) Show the structure of the Tree after deleting the following key from the tree constructed above
21, 31, 20, 10, 7, 25, 42, 4

A.1. Order = 4 max. keys = 4 − 1 = 3

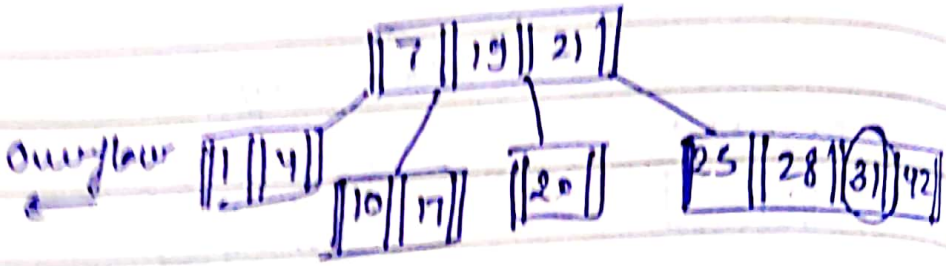


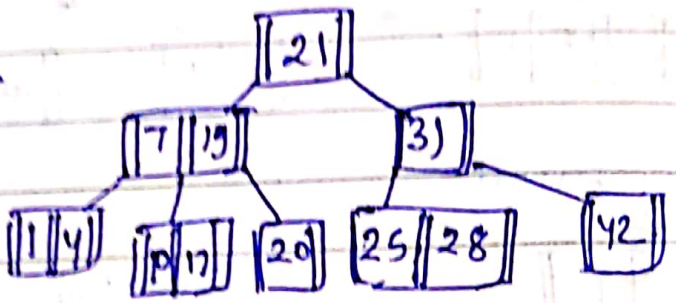
Scanned with CamScanner

Insert 28 →



↓ Insert 42





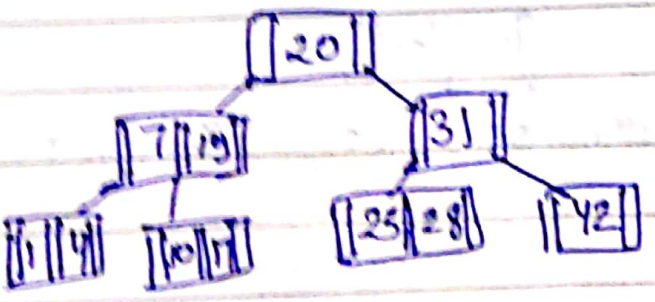
② Delete 21, 31, 20, 10, 7, 25, 42, 4

Underflow key in Node < $\lfloor \frac{n}{2} \rfloor = \lfloor \frac{4}{2} \rfloor = 2$

<2

Delete 21 Take inorder predecessor of 21 ~~to 21~~

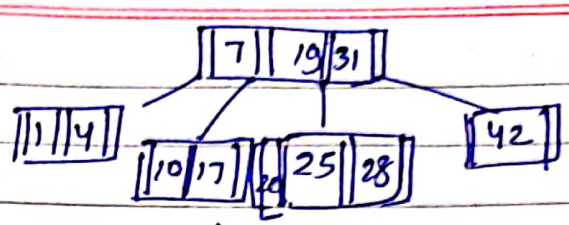
swap(21, thro 2) Delete 21



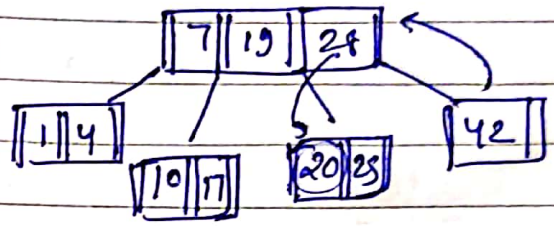
Delete 31 →

Min no. of child

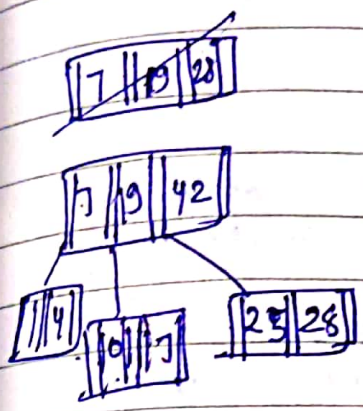
Delete 21 Merge left & Right



Delete 31



Delete 20



Delete 10

01.12.2022
Thursday.

Repeatable Read

T_1	T_2
write (x)	
	read (x)
write (x)	
	read (x)

ACID Properties:

$(Amount)_i = (Amount)_f$

- A — Atomicity → either complete transaction or nothing (rolled back)
- C — Consistency → T_1 and T_2 should not affect each other
- I — Isolation
- D — Durability ← Effect of commited transaction should be hold

System Log: Contains Transactions done in order. → Recovery purpose

Error / failure of commit
then reexecute the log file (failures)

Backup of the data / transaction

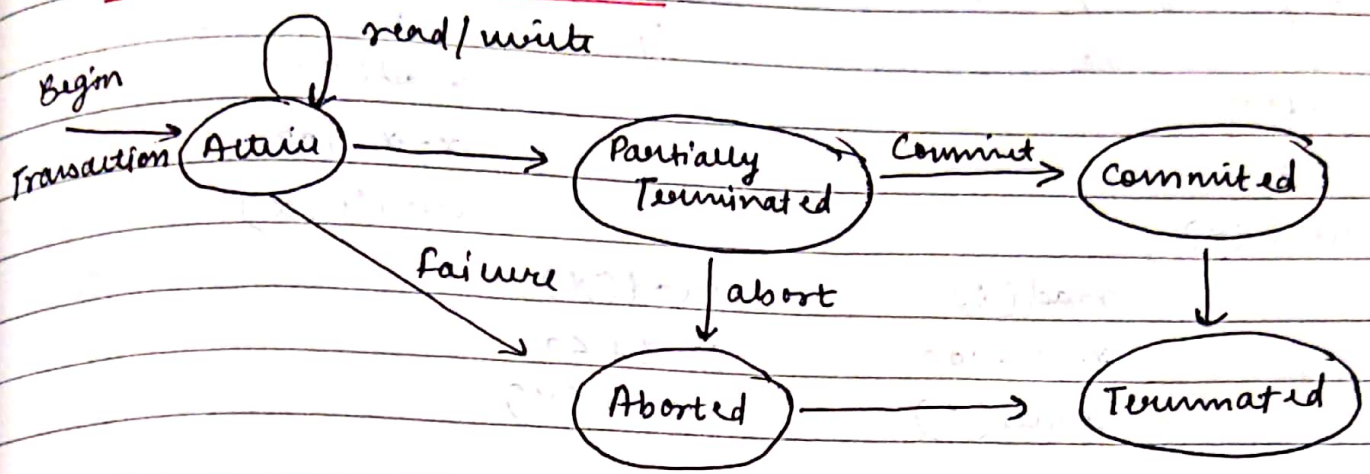
Happens frequently.

If error is going to come then do this explicitly.

undoing / Redoing can be done only with the help of Log file

Scanned with CamScanner

States of a Transaction



Serializability :-

Serial schedules → always correct / consistent database but response time ↑ses

To ↓se response time we use

Concurrent Schedules

↳ that CS which gives results = Result(serial) are serializable schedules

Serializable schedules ~~that~~ are those CS that are equivalent to some serial schedules and concept is serializability.

Two types of serializability :

1. conflicting serializability
2. view serializability

Scanned with CamScanner

① conflicting operations → RW Read write
→ WW Write write

T_1	T_2
read(X)	
X=X+50	
write(X)	
	read(X)
	X=X+100
	write(X)

T_1	T_2
	read(X)
	X=X+100
	write(X)
read(X)	
X=X+50	
write(X)	

RW T_1 Reads X from old — T_2 —||— old
T_2 Reads X from updated T_1 — T_1 —→ T_2

WW Final value by T_2 — Final value by T_1

If both have different variables then OK.
RR WR is also OK

T_1	T_2
	read(Y)
	Y=Y+50
	write(Y)
read(X)	
X=X-50	
write(X)	
read(Y)	
Y=Y+100	
write(Y)	

Serial

≡

T_1	T_2
read(X)	
X=X-50	
write(X)	
	read(Y)
	Y=Y+50
	write(Y)
read(Y)	
Y=Y+100	
write(Y)	

Swapping Non conflicting operation

Concurrent schedule

Scanned with CamScanner

Relative order of conflicting operations should be same in CS & SS.

⇓

Swapping Non conflicting so operation block

S_1

T_1	T_2
read(X);	
X = X - N;	
	write(X)
	X = X + M
write(X)	
read(Y)	
	write(X)
write(Y)	

S_2

T_1	T_2
read(X)	
X = X - N;	
write(X);	
Y = Y + N;	
write(Y)	
	read(X)
	X = X + M
	write(X)
	~~read(X)~~
	~~X = X - N~~

S_3

T_1	T_2
	read(X)
	X = X + M
	write(X)
	~~read(X)~~
	~~X = X - M~~
read(X)	
X = X - N;	
write(Y)	
Y = Y + N	
write(Y)	

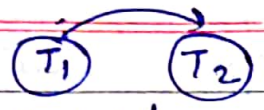
S_4

T_1	T_2
read(X)	
X = X - N	
write(X)	
	write(X)
	X = X + M
	write(X)
read(Y)	
Y = Y + N	
write(Y	

Scanned with CamScanner

No. of Nodes = No. of Transactions

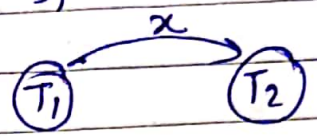
multi edge graph



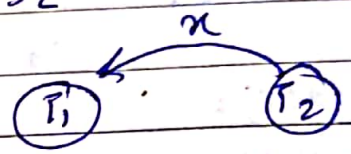
write(X)/
read(X)

write(X)/
~~read~~ write(X)

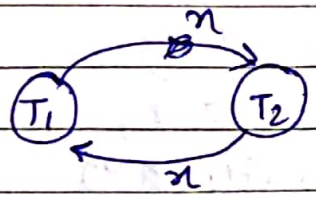
S_1



S_2



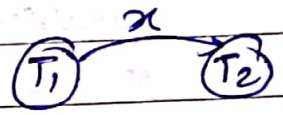
S_3



Cycle ∴ Can't be equal to some serial schedule

Not a Conflict Serializable

S_4



conflict serializable

Equivalent serial = Topological Order (linear order) just by viewing