22CST41 - Database Management Systems

CA-3 Answer key.

Part: A

1.

clustering Index	Non- clustering Index
→ Search key is in sequential order	→ Search key not in sequential order
→ Primary index	→ secondary index

2. Deficiencies of Static hashing
→ Fixed set of buckets
→ No way for database grow or shrink
→ Performance degradation
→ Expensive for real time

3. Bitmap Index

Gender:

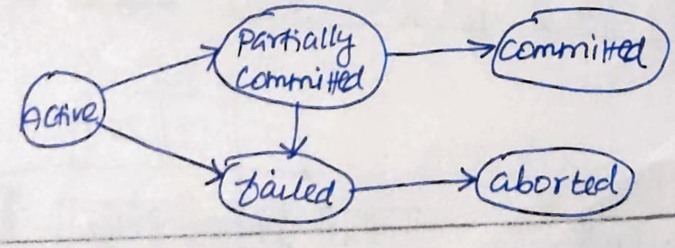
Record	0	1	2	3	4
Male	1	0	0	1	0
female	0	1	1	0	1

Salary:

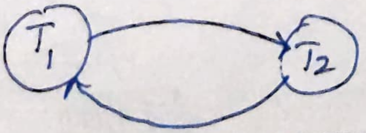
L1	10K to 20 K
L2	20K to 30 K
L3	30K TO 40 K
L4	40K to 50 K

Record	0	1	2	3	4
L1	1	0	1	0	0
L2	0	1	0	0	0
L3	0	0	0	1	0
L4	0	0	0	0	1

5. Transaction State:



4.



Since the precedence graph has cycle, it is not conflict serializable

6. Lock-compatibility Matrix:

	S	X
S	True	False
X	False	False

7. Thomas Write Rule:

→ Modified version of timestamp ordering protocol in which obsolete write operations are ignored when $TS(T_i) < W\text{-}timestamp(Q)$ instead of rollback.

8. Rollback:

ⓐ Total rollback: Abort the entire transaction & restart it

ⓑ partial rollback: Rollback the victim transaction alone.

9. Fuzzy Checkpoint:

→ Temporarily stop all updates by transactions

→ write a <checkpoint L> log record & force log to stable storage

→ note list M of modified buffer blocks

→ permit transactions & output to disk all modified buffes

10. Logical Undo Logging:

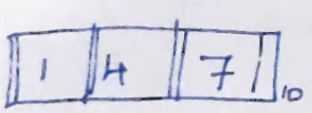
Logical undo: Insertion (resp deletions) are undone by executing a deletion (resp. insertion)

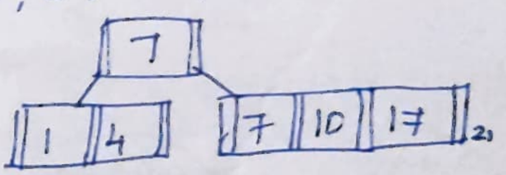
Logging of logical undo operation is called logical undo logging.

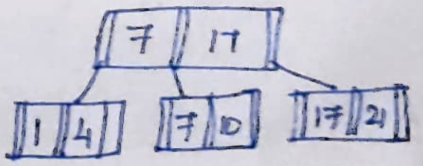
Part: B

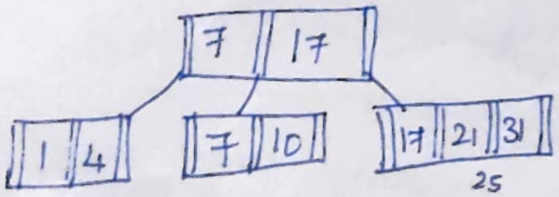
11. B^+ Tree → Order 4

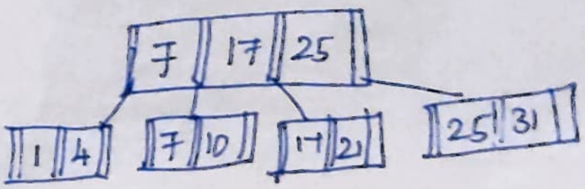
1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

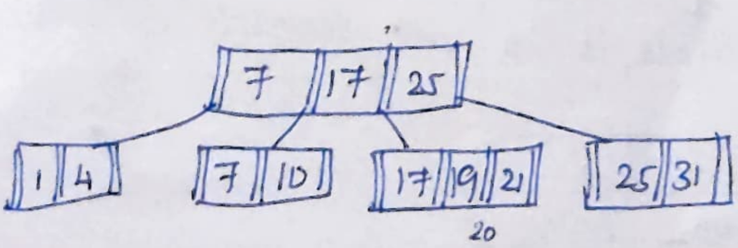


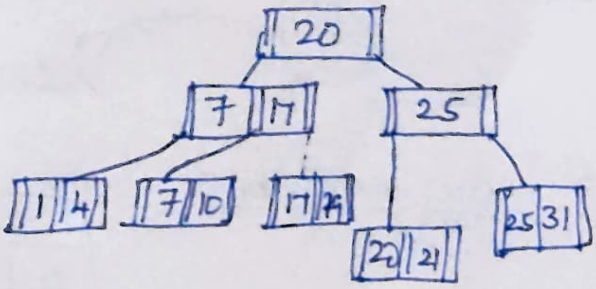


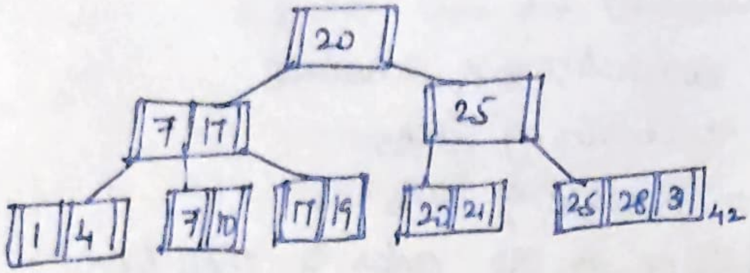


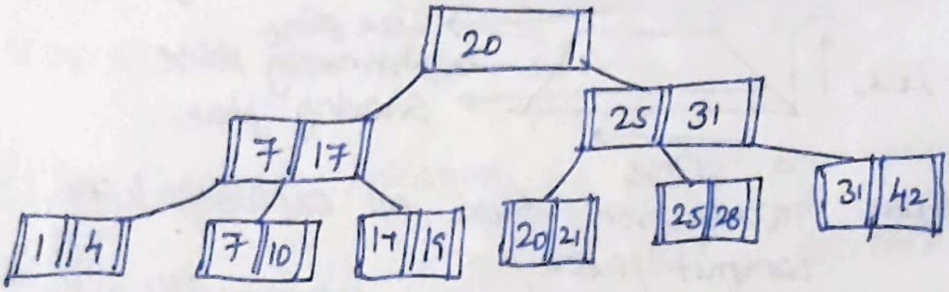












12.

S1: $R_1(x)$, $R_1(y)$, $R_2(x)$, $R_2(y)$, $w_2(y)$, $w_1(x)$

S2: $R_1(x)$, $R_2(x)$, $R_2(y)$, $w_2(y)$, $R_1(y)$, $w_1(x)$.

S1:

T_1	T_2
$R_1(x)$	
$R_1(y)$	
	$R_2(x)$
	$R_2(y)$
	$w_2(y)$
$w_1(x)$	

S2:

T_1	T_2
$R_1(x)$	
	$R_2(x)$
	$R_2(y)$
	$w_2(y)$
$R_1(y)$	
$w_1(x)$	

Conflict: Instructions:

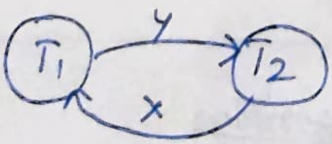
$R_1(y), w_2(y): T_1 \to T_2$

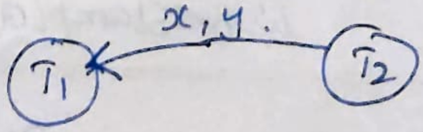
$R_2(x), w_1(x): T_2 \to T_1$

$R_2(x), w_1(x): T_2 \to T_1$

$w_2(y), R_1(y): T_2 \to T_1$

Precedence graph:





→ Schedule S_1 has cycle hence it is not conflict serializable

→ Schedule S_2 has not contain cycle hence it is conflict serializable.

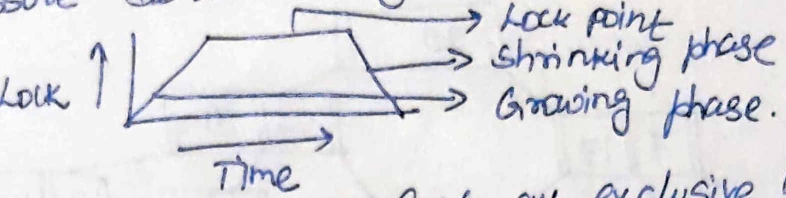
(13) (i) Two phase Locking protocol

→ protocol ensure conflict serializable schedules

→ phase 1: Growing phase → obtaining locks

→ phase 2: Shrinking phase → Release of locks

→ Protocol assure serializability in the order of their lock points.



→ Strict 2 phase: Transaction hold all exclusive locks till it commit / abort.

→ Rigorous 2 phase: Transaction hold all locks till it commit / abort.

(13) (ii) Timestamp based protocol

→ Each transaction has unique timestamp when it enter into system

→ Timestamp could be based on a logical counter

→ It manage concurrent execution such that timestamp order = serializability order.

→ For T_i issues Read(Q)

$TS(T_i) \leq W\_timestamp(Q) \rightarrow$ rollback

$TS(T_i) > W\_timestamp(Q) \rightarrow$ Read

} $R\_timestamp(Q) = Max(R\text{-}timestamp(Q), TS(T_i))$

→ For T_i issues Write(Q)

$TS(T_i) < R\text{-}timestamp(Q) \rightarrow$ rollback

$TS(T_i) < W\text{-}timestamp(Q) \rightarrow$ rollback

else Read write

$W\_timestamp(Q) = TS(T_i)$

(14). Recovery Algorithm

It has two parts.

1. Action taken during normal transaction process to ensure enough information exist to recover from failure

2. Action taken after failure to recover the database content to ensure atomicity, consistency & durability.

Log based Recovery:

→ Log is sequence of log records

→ Ex: <T_i Start> T_i Started

<T_i Commit> T_i committed

<T_i Abort> T_i aborted

Log record: <T_i , X_i , V_1 , V_2> → new value

Transaction id   Data item id   old value

Checkpoint:

→ Streamline recovery procedure by periodically performing checkpoint.

⇒ Log record <checkpoint L> → L is list of all transactions active at time.

Recovery Algorithm:

Two phases

Redo phase:

1. Find last <Checkpoint L> & set to undo-List to L
2. Redo <T_i, X_j, V_2> to <T_i, x_j , X_i>
3. <T_i Start> ⇒ add T_i to undo list
4. <T_i commit> or <T_i Abort> ⇒ Remove T_i from undo list

Undo phase:

1. Scan log backwards from end
2. Redo V_1 to x_j , write <T_i , x_j , V_1>
3. write <T_i abort> remove T_i from undo list
4. Stop when undo list is empty.