SACHIN DUHAN
2K17/MC/087

DBMS ASSIGNMENT-3

Q.1 ——————?

(i)

T_1

BEGIN
R(X)
R(Y)
W(X)
COMMIT

T_2

BEGIN
R(X) → 10.
R(Y)
W(X) → 13
W(Y)
ABORT.

(T_1 is reading data of X written by T_2 before committing).

(ii)

T_1

BEGIN
R(X)
R(Y)

W(X)
COMMIT

T_2

BEGIN
R(X)
R(Y)
W(X)

W(Y)
COMMIT

(iii)

T_1

BEGIN
R(X)
R(Y)

W(X)
COMMIT

T_2

BEGIN
R(X)
R(Y)

W(X)
COMMIT.

Scanned with CamScanner

iv) If we use strict 2PL the locking and Unlocking will be done in 2 phases and all exclusive locks will have held till the transaction commits or aborts and shared locks. can be released anytime during the second phase. If we apply ~~this~~ strict. 2PL. only ~~stab~~ serializable schedule will be allowed and all the 3 schedule listed above will be shared by S_1 and release of lock by U_1 and commit by C. So following will be execution. of schedules wherere strict 2PL will ensure stability by not granting locks which may create conflicts.

(i) $S_2(X), R_2(X), S_2(Y), R_2(Y), X_2(X), W_2(X), S_1(X)$ - request not granted, $X_2(Y), W_2(Y), U_2(X), U_2(Y), S_1(X), R_1(X) S_1(Y),$
$R_1(Y), X_1(X), W_1(X), U_1(X), U_1(Y), C_1$

(ii) $S_2(X), R_2(X), S_2(Y), R_2(Y), X_2(X), W_2(X), S_1(X)$ - request not granted, $X_2(Y), W_2(Y), U_2(X), U_2(Y), C_2, S_1(X), R_1(Y), S_1(Y), R_1(Y)$
$X_1(X), W_1(X), U_1(X)$,

(iii) $S_2(X), R_2(X), R_2(Y), S_1(X)$ - request not granted, $X_2(X), W_2(X)$
$X_2(Y), W_2(Y), W_2(Y), U_2(X), U_2(Y), C_2, S_1(X), R_1(X)$
$S_1(Y), R_1(Y), X_1(X), W_1(X), U_1(X), U_1(Y), C_1$.

U'2 — — — — ?

(i) · S(A) at t_2 : g
S(B) at t_3 : g
S(C) at t_4 : g
X(B) at t_5 : b.
S(A) at t_6 : b
S(C) at t_1 : g

(ii)

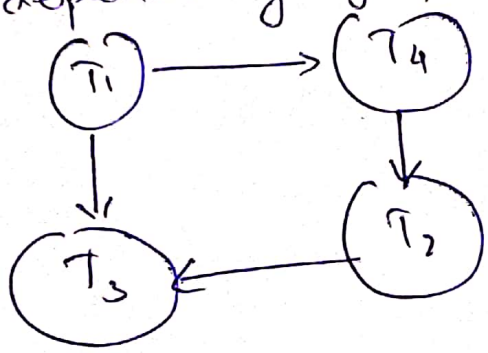
(iii) A deadlock doesnot exists because there is no cycle in the dependency graph.

Scanned with CamScanner

Q.3 ——— ——— ———?

(i)
- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : b
- X(D) at t_5 : g
- X(C) at t_6 : b
- X(B) at t_1 : b.
- S(D) at t_8 : b.

(ii) A deadlock exists because there is a cycle in the dependency graph.



(iii)
- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : a (T_1 aborts)
- X(D) at t_5 : g
- X(C) at t_6 : a (T_2 aborts)
- X(B) at t_7 : g (No locks held on B since t_1 was aborted)
- S(D) at t_8 : g (No locks held on D since T_2 was aborted)

(iv)
- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : b
- X(D) at t_5 : g
- X(C) at t_6 : b
- X(B) at t_7 : g (T_1 Aborts)
- S(D) at t_8 : g (T_2 Aborts)

Scanned with CamScanner

Q.4 — — — ?

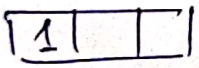
(a) (III) X(A) at t_2: g
S(C) at t_3: g
S(A) at t_4: a (T_1 aborts)
X(D) at t_5: g.
X(C) at t_6: a (T_2 aborts).
X(B) at t_7: g (No locks held on B since T_1 was aborted).
S(B) at t_8: g (No locks held on D since T_2 was Aborted).

(IV) X(A) at t_2: g
S(C) at t_3: g
S(A) at t_4: b
X(B) at t_5: g
X(C) at t_6: b
X(B) at t_7

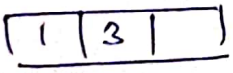
Q.4 — — ?

(a) B tree.

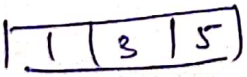
Insert 1



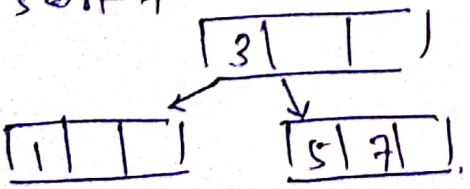
Insert 3



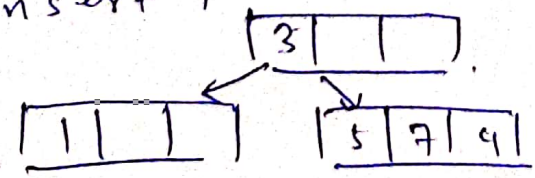
Insert 5



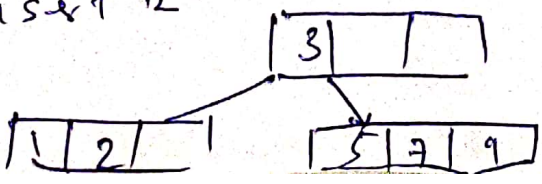
Insert 7



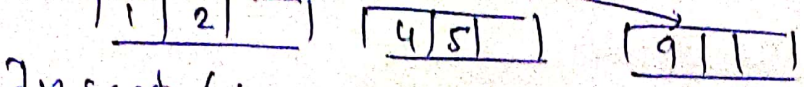
Insert 9



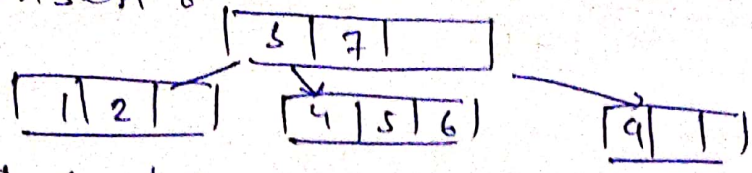
Insert 2



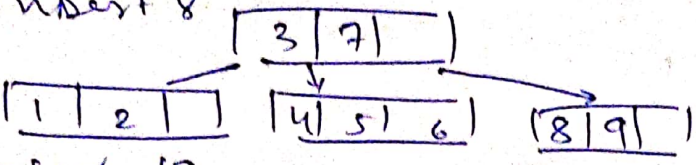
Scanned with CamScanner



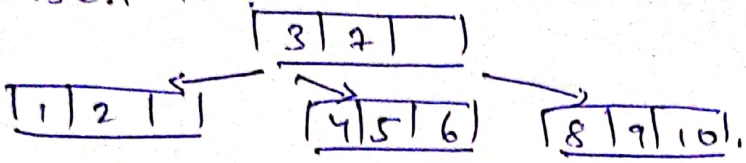
Insert 6.



Insert 8

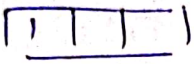


Insert 10

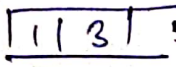


B+ tree:

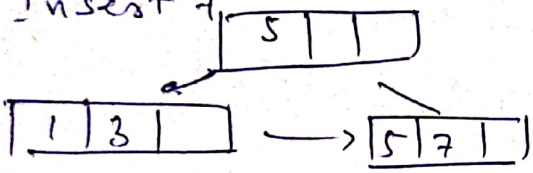
Insert 1



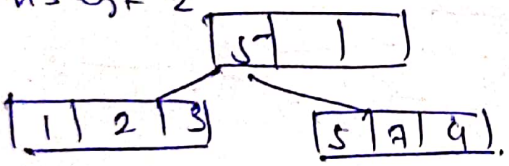
Insert 3,5



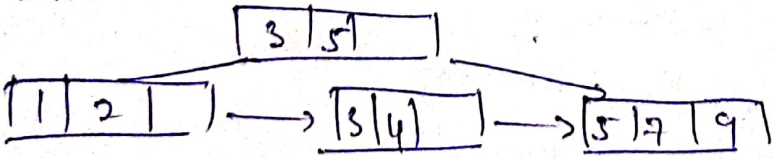
Insert 7



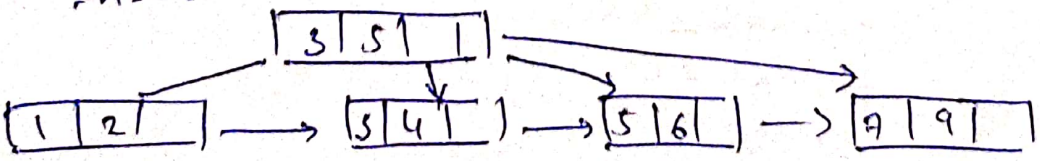
Insert 2



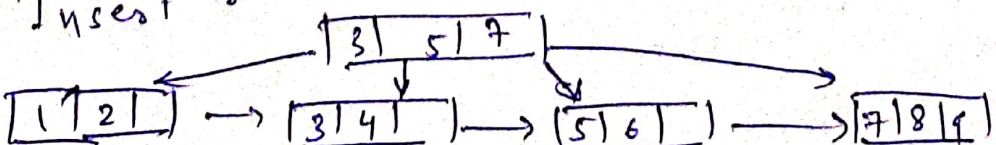
Insert 4.



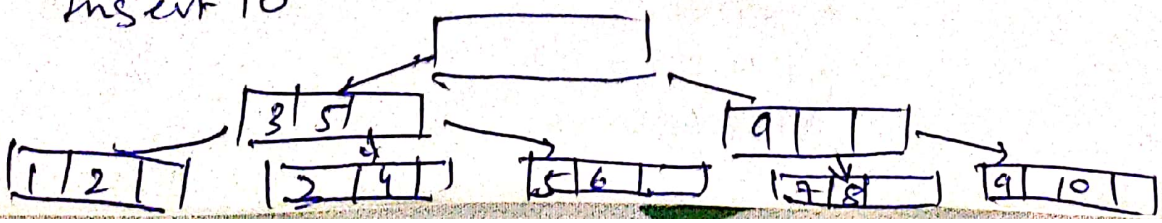
Insert 6.



Insert 8



Insert 10

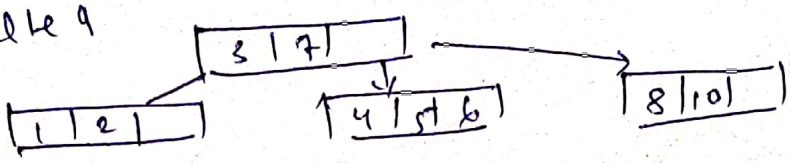


Scanned with CamScanner

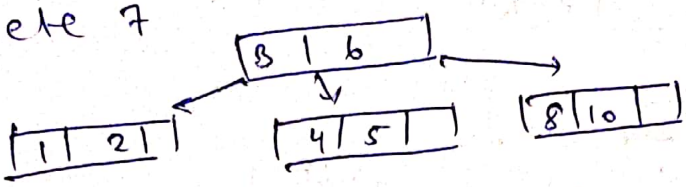
b) Delete 9, 7, 8

B tree.

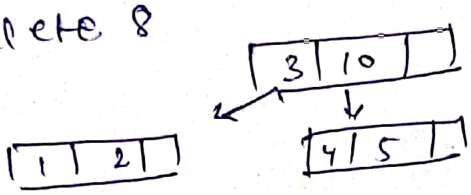
Delete 9



Delete 7

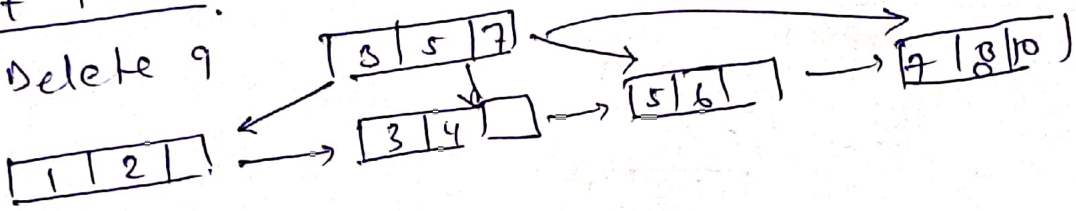


Delete 8

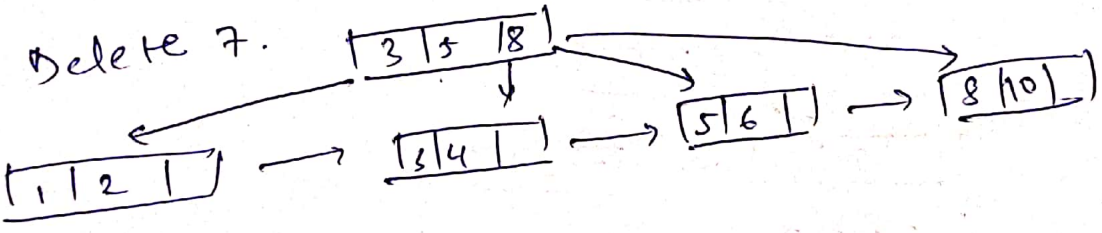


B+ tree.

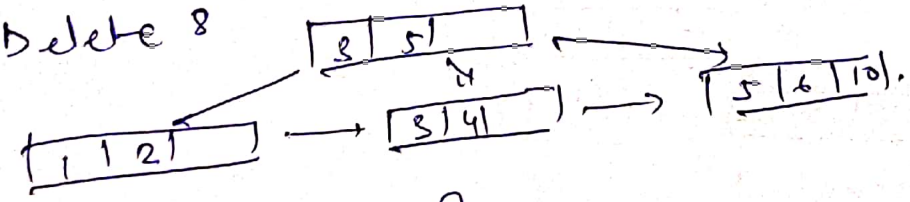
Delete 9



Delete 7.



Delete 8



— ?

Q.5 — — — — — —

(ii) Converting into binary form.

000001 → 1

101000 – 40.

111101 – 61

000010 – 2

000100 – 4

001111 – 15

011110 – 30

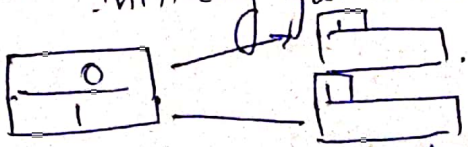
010001 – 9.

010100 – 20

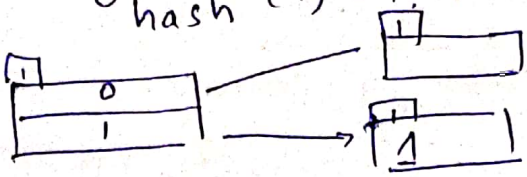
011010 – 26

Scanned with CamScanner

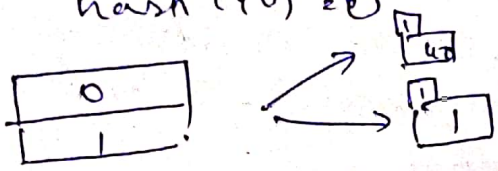
Initially global and local depth is 1.



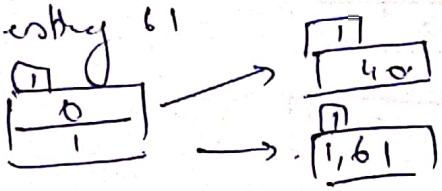
Inserting 1, global depth is 1
hash (1) = 1.



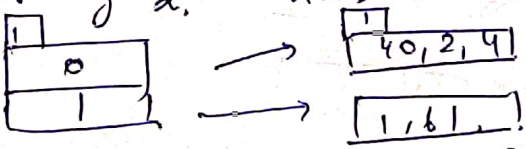
Inserting 40
hash (40) = 0



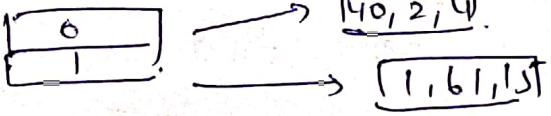
Inserting 61



Inserting 2. hash (4) = 0.

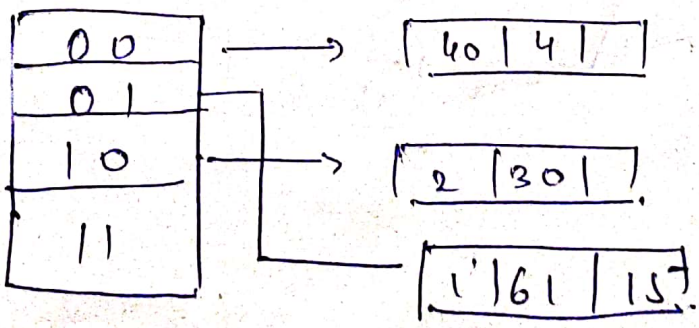


Inserting 15 hash (15) = 1.



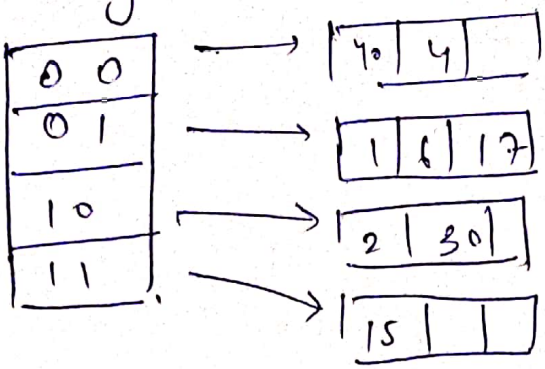
Inserting 30, hash (30) = 0 => overflow!
Directory expansion. & rehashing :) all the numbers inserted upto now occurs.
global depth = 1 + 1 = 2.

hash (1) = 01 ; hash (61) = 01
hash (40) = 00 hash (2) = 10, hash (4) = 00
hash (15) = 11 hash (30) = 10,



Scanned with CamScanner

Inserting 17 hash(17) = 01 OVERFLOW!

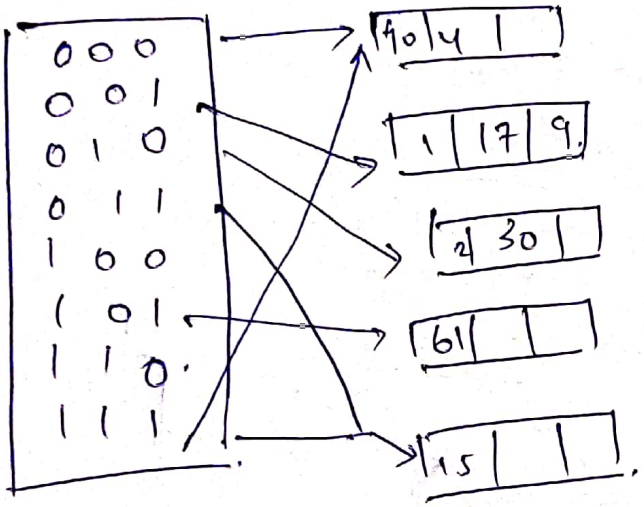


Inserting 9 hash(9) = 01 OVERFLOW.

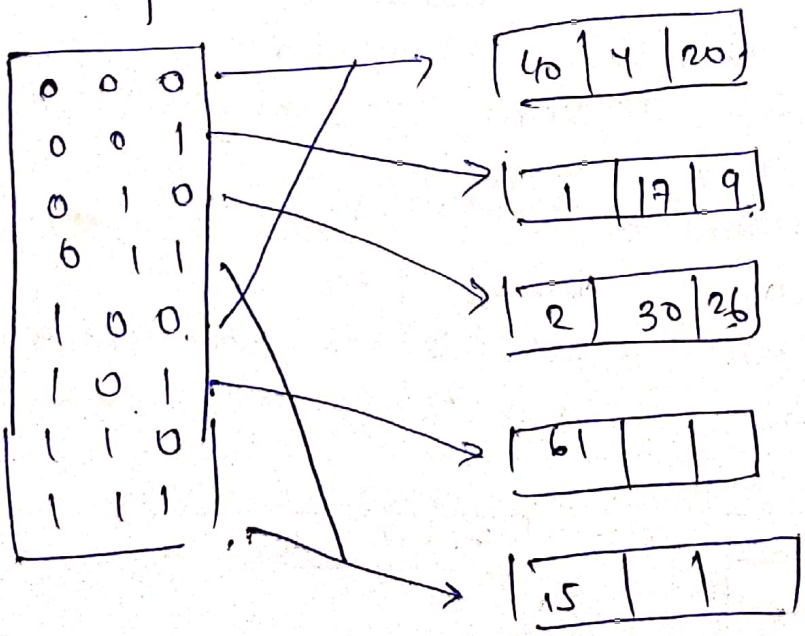
local Depth = global depth.

⇒ directory expansion & rehashing

hash(1) = 001, hash(61) = 101, hash(17) = 001, hash(9) = 001



Inserting 26 hash(26) = 010.



Scanned with CamScanner

Q.6. — — — ?

(a) In DBMS, hashing a technique to directly search the location and desired data disk without using index structure. Data is stored in form of data bloder whose adress is generated by applying a hash function in memory location where the records are stored known as data block or data bucket.

(b) 1) chaining
It is a method in which additional field with which data is introduced.

2) linear probing
In this, collosion can be solved by placing the second record linearly down, whenever empty place is found.

3) quadratic probing
hash(key, i) = (hash(k) + i^2) mood m

(c)

STATIC HASHING	DYNAMIC HASHING
A HASHING technique that allow users to perform lookups on a finalized dictionary (all objects in dictionary are final and not changing)	A hashing technique in which data buckets are added and removed dynamically and on demand.
Resultant data bucket adress is always the same.	Data buckets change depending on the record.
less efficient.	more efficient.

Scanned with CamScanner