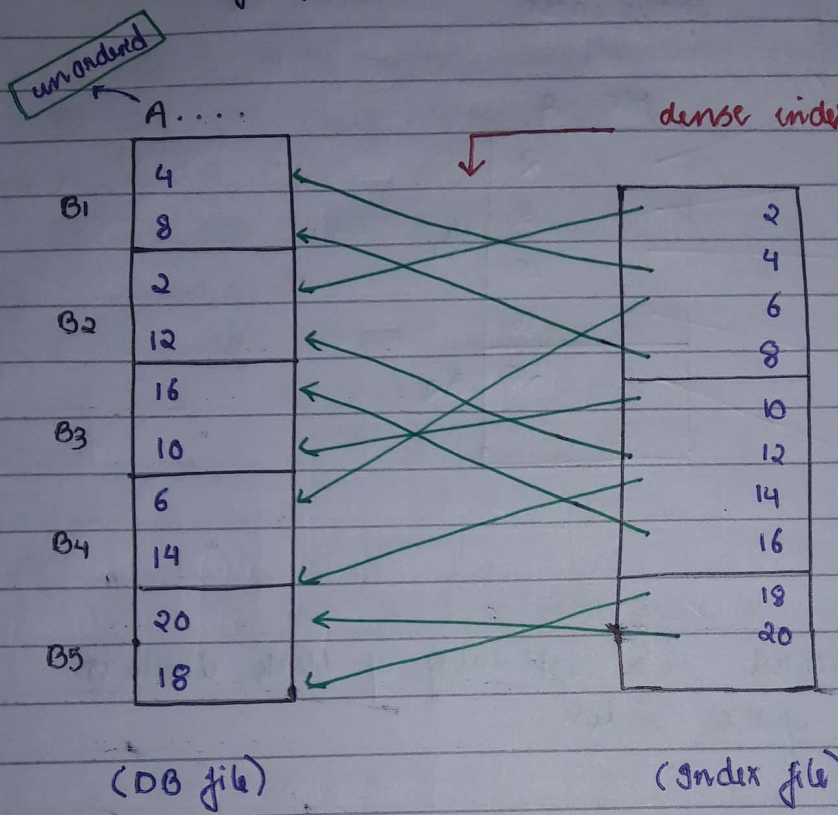
- For each record of DB file, there exist an entry in index file.

Indexing for unordered field of DB file —



No. of index entry = No. of records

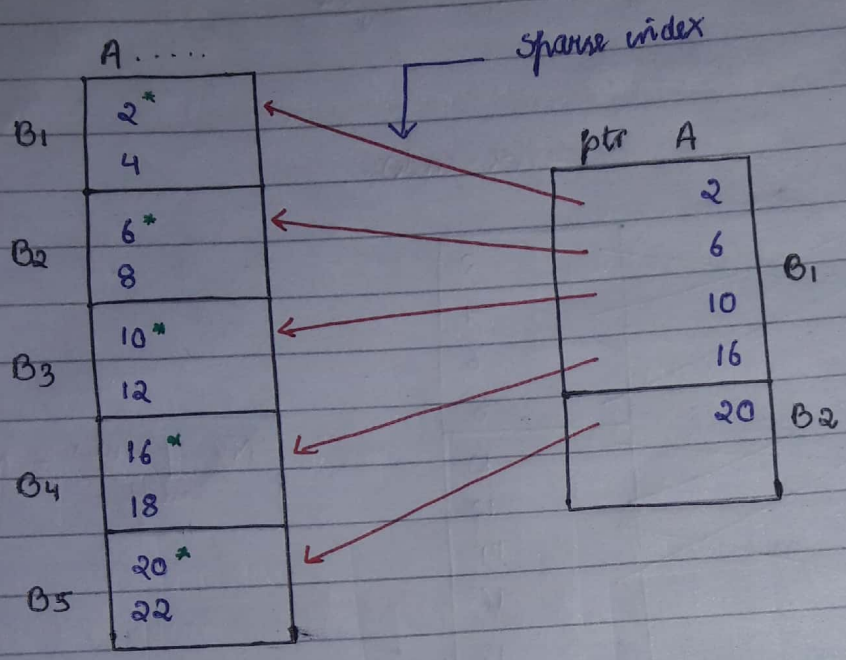
- Dense indexing can be done for ordered as well as unordered field.

Sparse Index —

→ Less entries in index file. (objective)

→ For set of record there exist an entry in index file (i.e. for many ~~entry~~ record there is one entry)

- sparse index is possible only if search key is ordered field of the DB file.



* denote anchor record, i.e 1st entry of block which is used as record of sparse index

For sparse index,

No. of index file entries < No. of records of DB files.

but, by default,

No. of index file entries ≅ No. of blocks of DB files.

Ques: File consists 50,000 records with record size 100 Bytes
Block size = 1024 Bytes
search key = 10 Bytes
Pointer = 5 Bytes.

** ① No. of dense index blocks ?
② I/O cost using dense index ?
** ③ No. of sparse index blocks ?
④ I/O cost using sparse index ?

⑤ I/O cost without index
a) Based on ordered field ?
b) Based on unordered field ?

Solⁿ:

No. of dense index block = $\frac{\text{No. of entries}}{\text{Block size}}$.

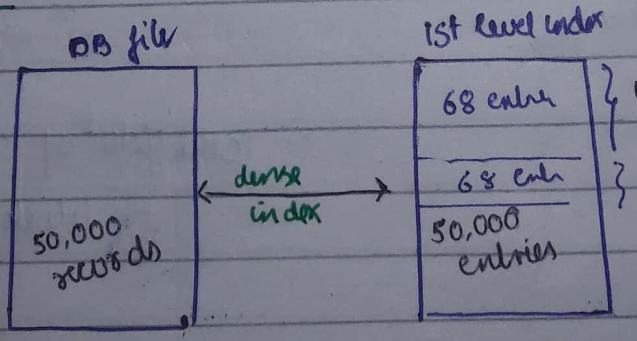
$= \frac{50,000 \times 15}{1024}$ B ⇒ wrong answer (as all 1024 block cannot be used because of unspan)

X it gives

1024 → $\frac{1024}{15}$

⇒ 68
68 × 15 = 1024
i.e 4B is unused

dense index —



Block factor of index = $\left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024-0}{10+5} \right\rfloor$
= 68 entries/block

$\therefore$ No. of dense index blocks $= \left\lceil \frac{50,000}{68} \right\rceil$

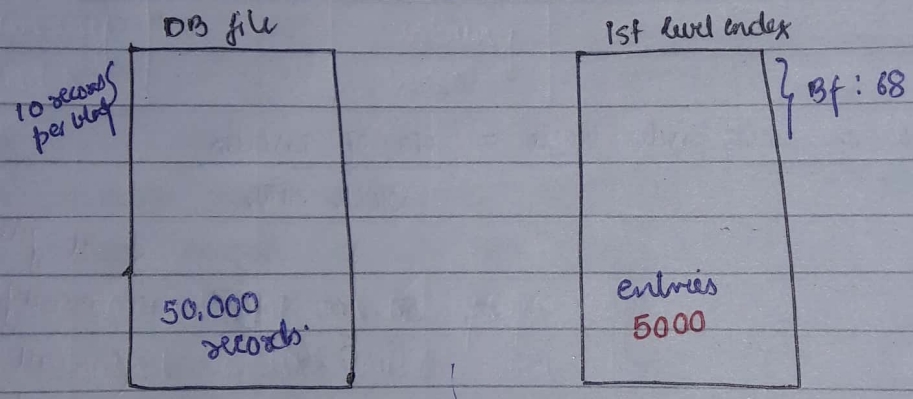
$= 736$ blocks Ans

2) I/O cost using dense index $= \lceil \log_2 m \rceil + 1$

$= \lceil \log_2 736 \rceil + 1 \Rightarrow 10 + 1$

$\Rightarrow$ 11 blocks.

Sparse index block –



Block factor of DB $= \left\lfloor \frac{B-H}{R} \right\rfloor$

$= \left\lfloor \frac{1024-0}{100} \right\rfloor$

$= 10$ record/block

Block factor of indexed $= \left\lfloor \frac{B-H}{K+P} \right\rfloor$

$= \frac{1024-0}{10+5}$

$= 68$ entries/block

$\therefore$ Total block $= \frac{50,000}{10}$

$= 5000$.

$\therefore$ Total No. of sparse index block $= \left\lceil \frac{5000}{68} \right\rceil$

$= 74$ blocks.

Ans

4) I/O cost using sparse index = $\lceil \log_2 m \rceil + 1$

$= \lceil \log_2 74 \rceil + 1$

$= 8$ blocks Ans

5) I/O cost without

a) based on ordered field).

(we can use sparse)

$= \lceil \log_2 n \rceil$

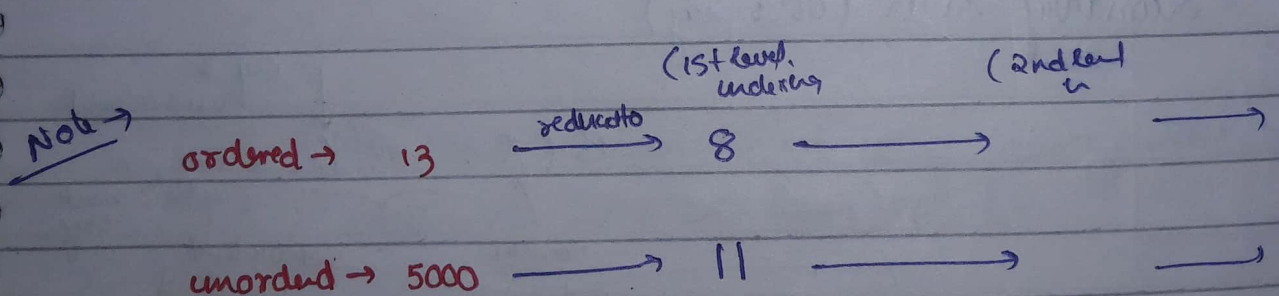
$= \lceil \log_2 5000 \rceil$

$= 13$ blocks Ans

b) based on unordered field?

$= n$

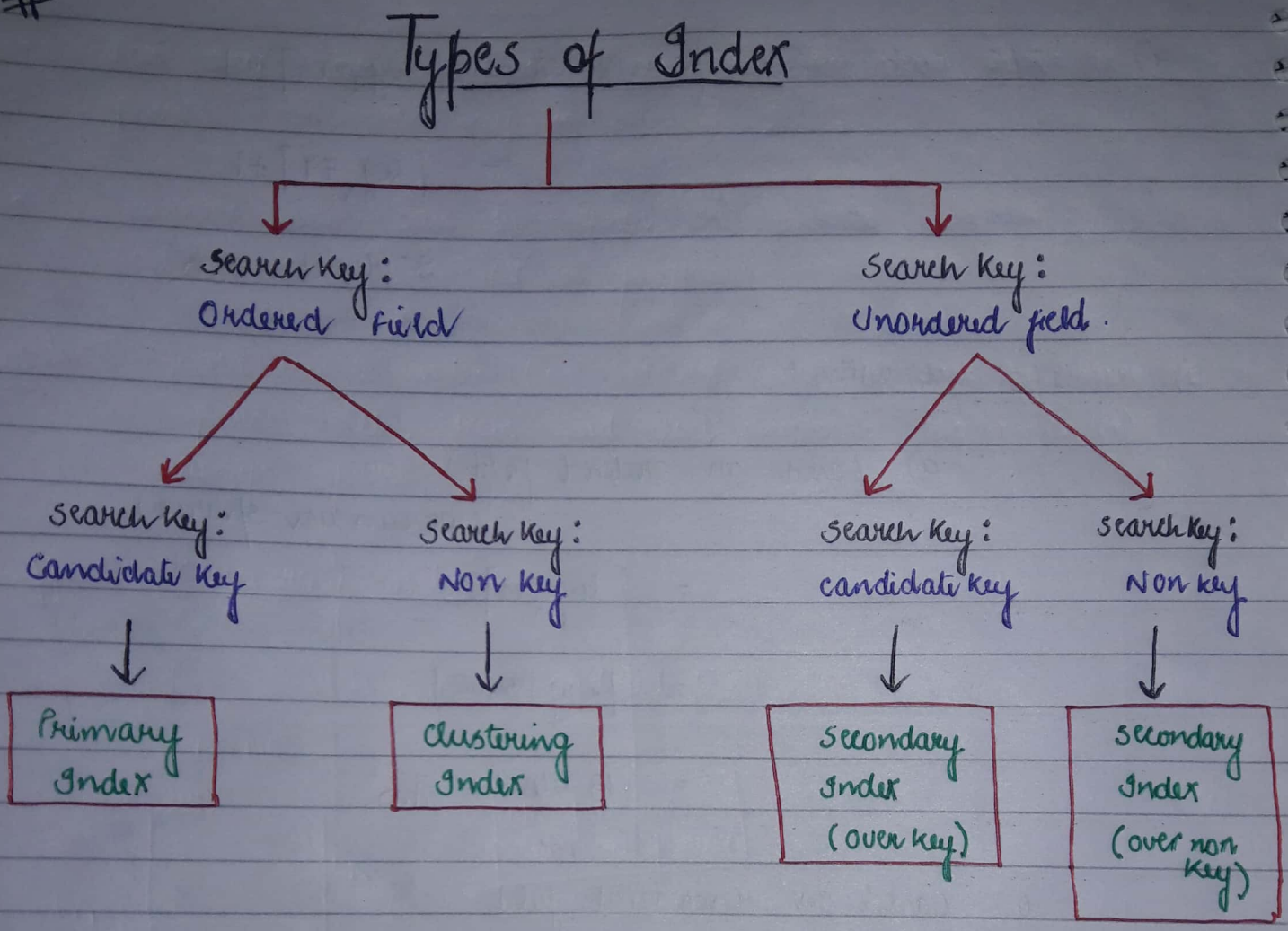
$= 5000$ blocks Ans



Search key : fields used for indexing DB file.

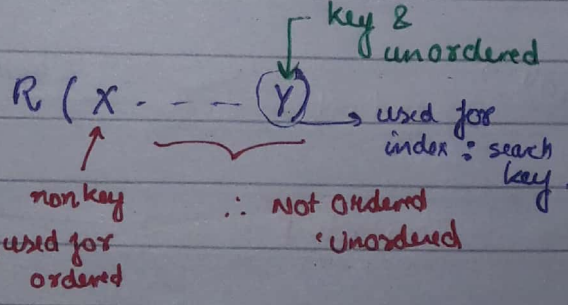
#

Types of Index



Ques: Records of rel R physically ordered over non key field X, and indx build over key field of rel R.

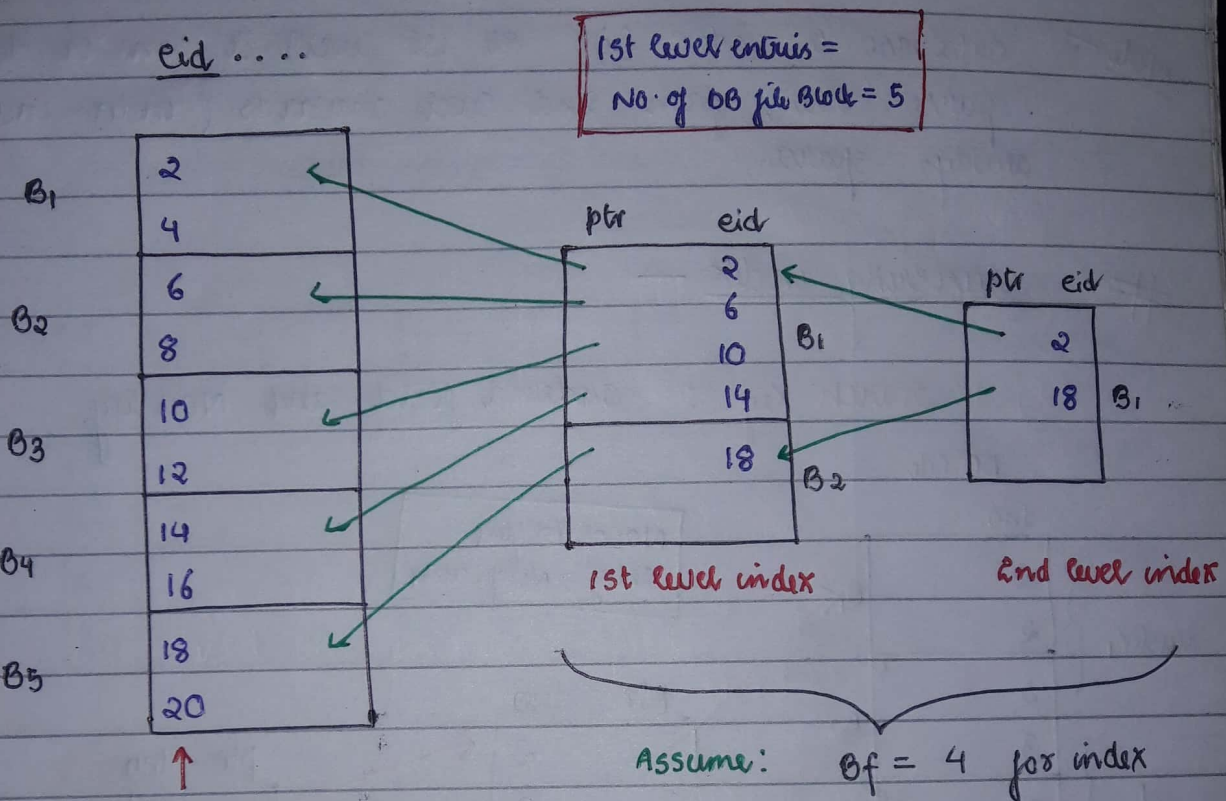
'Secondary Index (over key)'



Primary Index —

where search key is key and ordered field of DB file, it is primary index.

Eg —



Assume: (DB with B.f = 2)

I/O cost to access record using primary index with multilevel index : (k+1) blocks

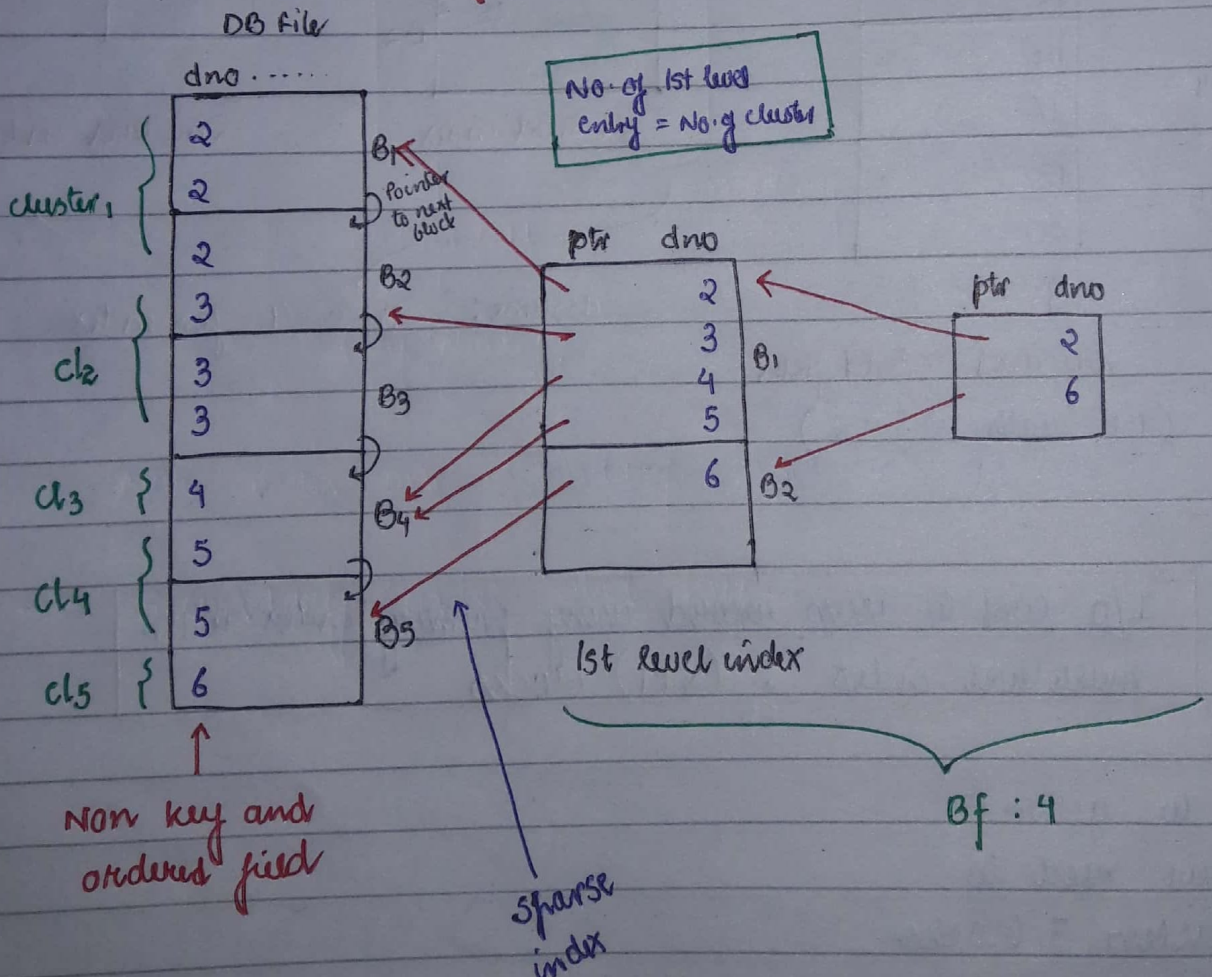
Here, to access it, we need to access 3 blocks.

- For any DB relation atmost one primary index is possible. (as search key is ordered field)
- Primary index can be dense or sparse. (because of ordered field), but sparse PI is preffered.

Note:- Database to index (i.e at 1st level) it can be dense or sparse but, from 2nd level onwards (index - index), always sparse.

Clustering index —

Search Key : ordered field and Non key

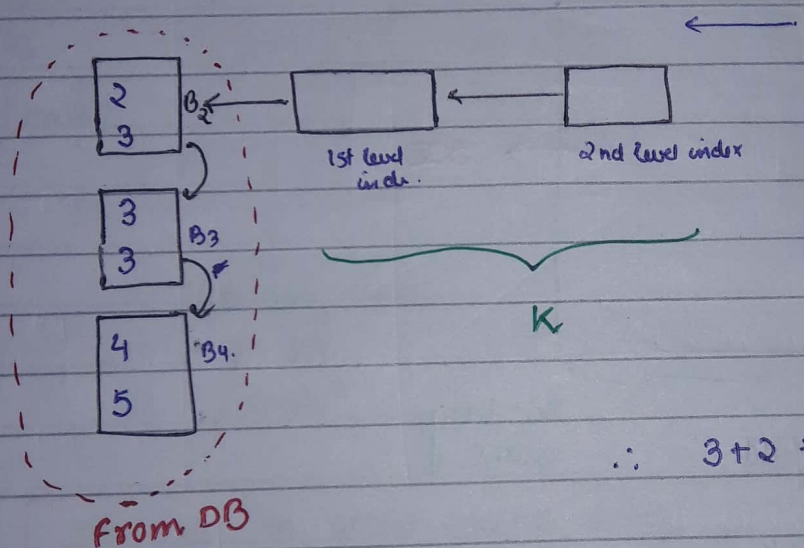


I/O cost to access cluster of records using clustering index at 1st level followed with multilevel indexing =

K + one or more DB blocks access until next cluster begin

Eg -

```
select * from emp
    where dno = 3;
```



It goes on sequentially until next cluster begins i.e $B_2 \rightarrow B_3 \rightarrow B_4$.

$\therefore$ 3+2 ⇒ 5

→ Clustering index is mostly sparse index. (clustering index can be dense if each cluster is with only 1 record)

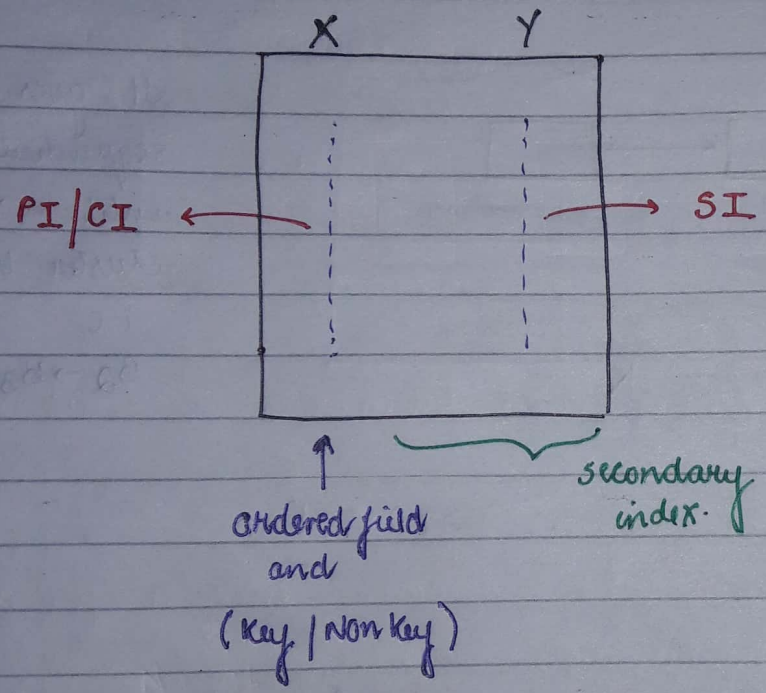
→ For any DB relation atmost one clustering index is possible (because search key is ordered field)

→ For any DB relation, either primary index or clustering index is possible but not both.

Secondary Index —

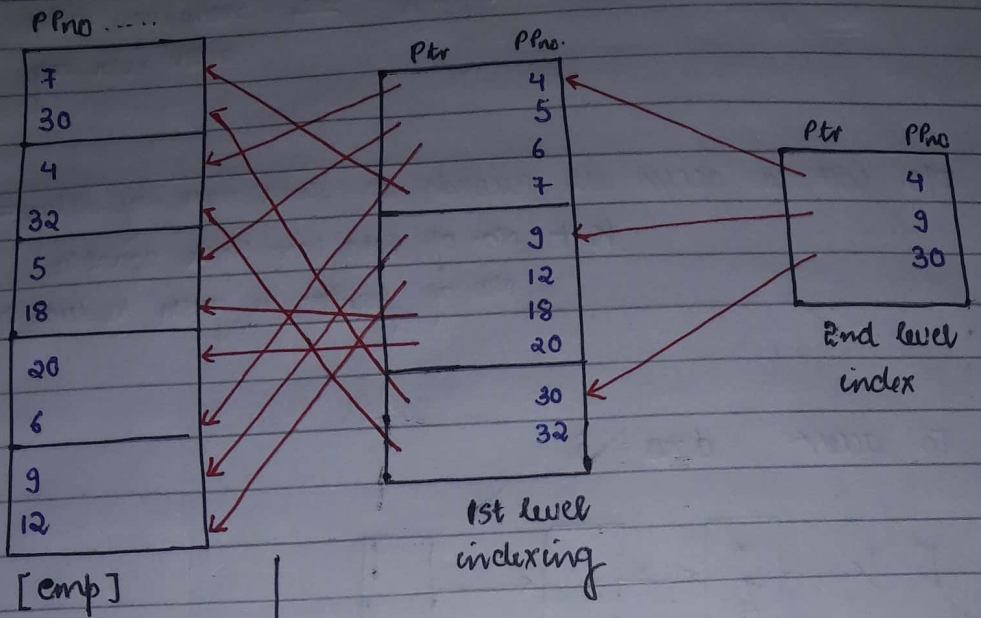
Search Key : unordered field
(and)
Key / Non Key

- It is called as secondary index as it is secondary possible way to access data even when primary index / clustering index already exist.



- More than one secondary index are possible.

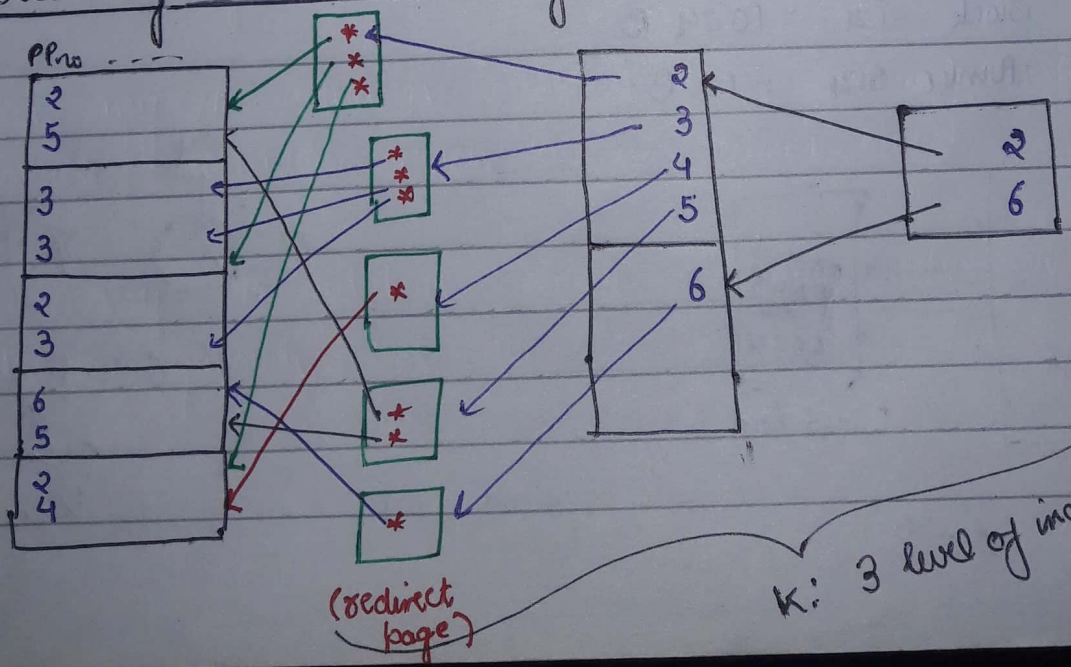
→ secondary index over key :



secondary index over key (dense

I/O cost to access record using SI with multi level indexing = (k+1) blocks

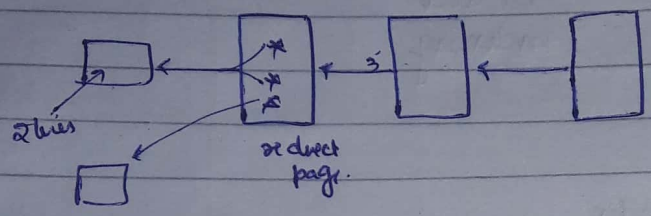
→ secondary index over non key:



Note - clustering index + redirect page ≃ secondary index over non key

- I/O cost to access all records of same non key value = K + No. of data base block equal to No. of pointers in given redirect page

Eg- To access dno = 3



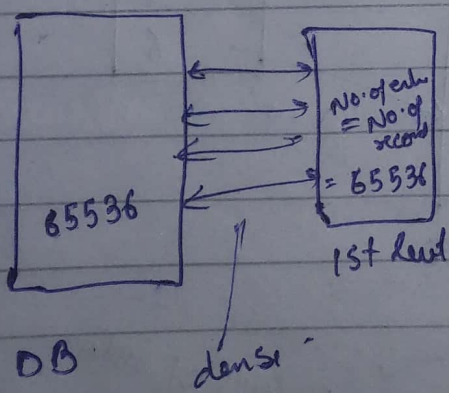
⇒ 3 + 3 pointers in redirect page

⇒ 6.

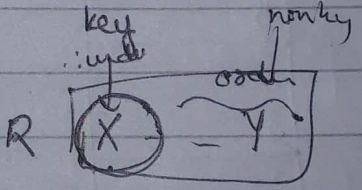
Pg 65 (11)

No of records = 65536
Record size = 32 B
search key size = 6 B
Block size = 1024 B
Pointer size = 12 B.

a)



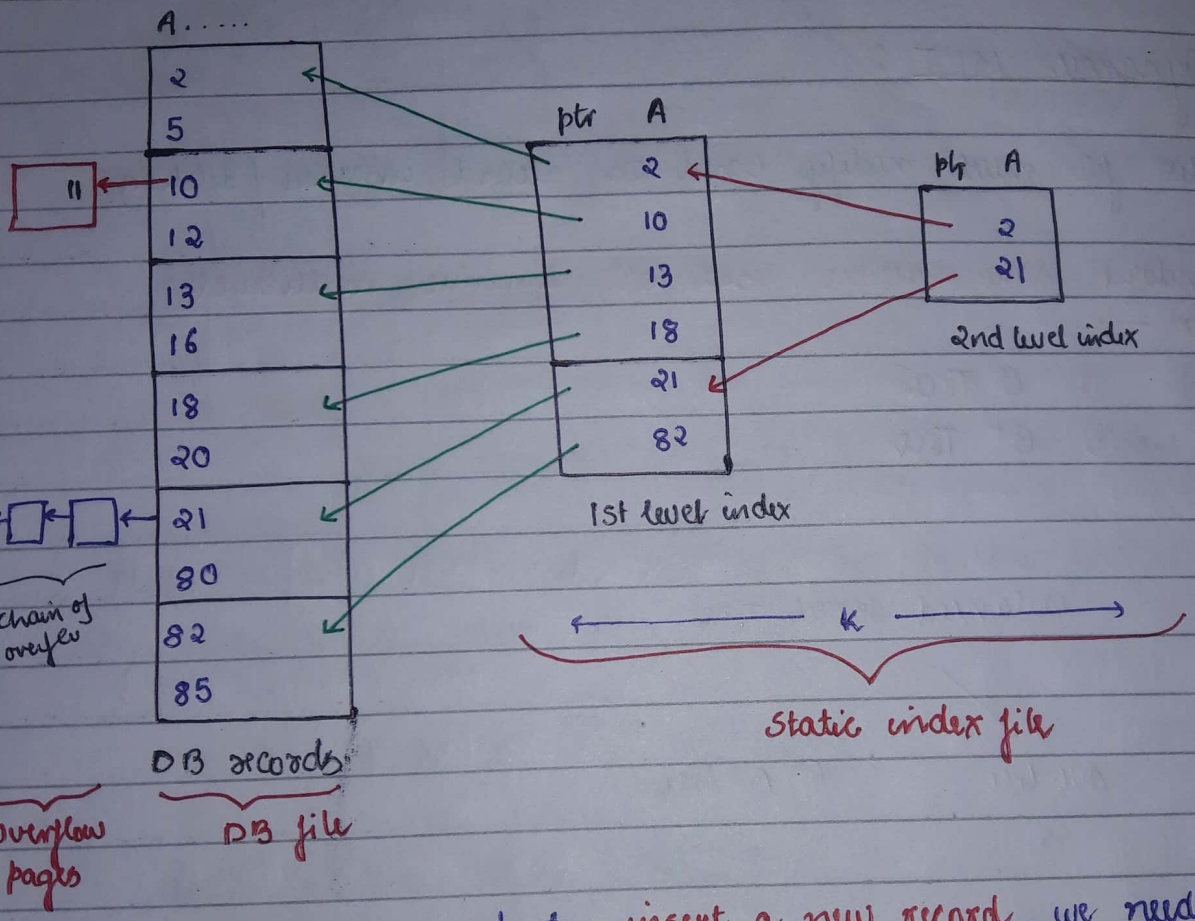
I/O cost = K+1



Secondary index over key.

Multilevel Indexing

① Static MLI : Index file is constant and newly inserted records are stored in overflow pages.



Now, when we need to insert a new record, we need to change the index completely. (which is complicated)

↓

So, we make index file static and add overflow pages

↓ but an issue arises

Access cost increases

Disadvantage of static MLI —

- Worst case access time = $O(n)$, where n is no. of record.
- Minimum usage of index block can become 0 %.

② Dynamic MLI :

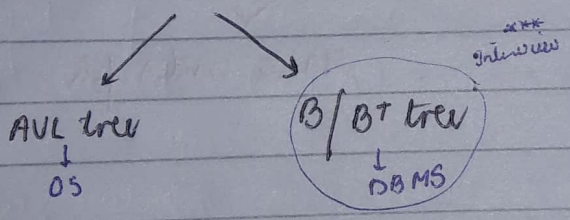
→ Index file must modify based on record insertion / deletion.

→ Standard Data structure used for dynamic multilevel index —

① B Tree

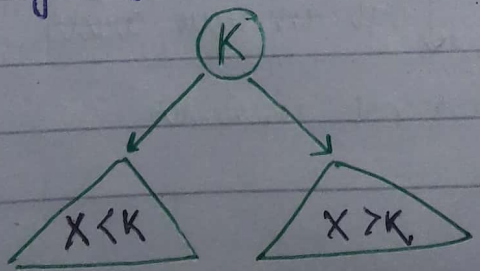
② B^+ Tree

Balanced search Tree



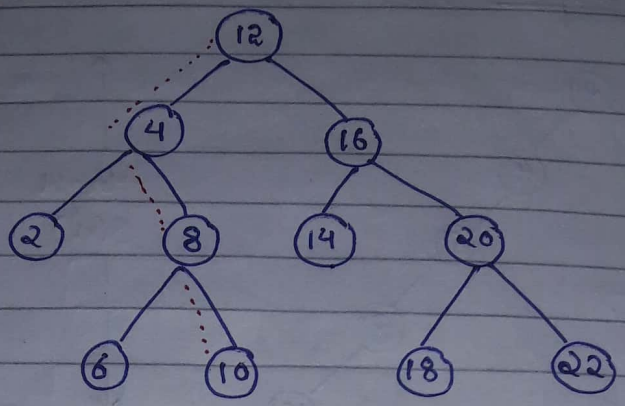
Search Tree —

Data structure used to store keys in tree format and every parent key is more than keys of left subtree and less than keys of right subtree



to perform search operⁿ efficiently.

Eg – BST



To search x = 10

At every level only 1 node is accessed

Probe sequence : 12, 4, 8, 10 (success)

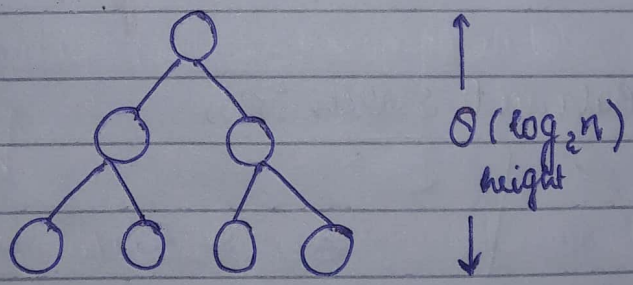
To search x = 17

probe sequence : 12, 16, 20, 18 (fails)

∴ Probe sequence to search any key in search tree is exactly one node access from each level.

For a search tree of 'n' key

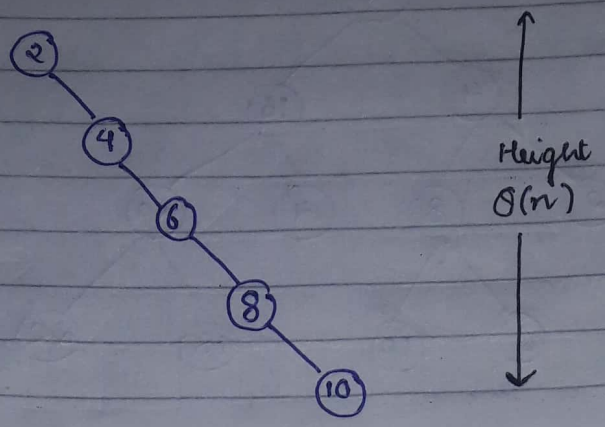
advantage a) Min height : $O(\log n)$



search cost = $O(\log n)$

disadvantage

6) Max height : O(n)



search cost = O(n)

To solve this problem, we have BST.

Binary search Tree

(Height Restricted search Trees)

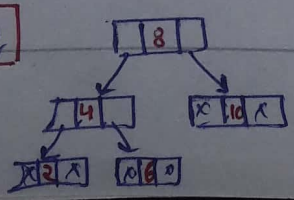
- Max height of search tree for n distinct keys should not exceed O(log n) → Objective of BST
- Advantage : worst case search cost to search the n distinct element is O(log n).

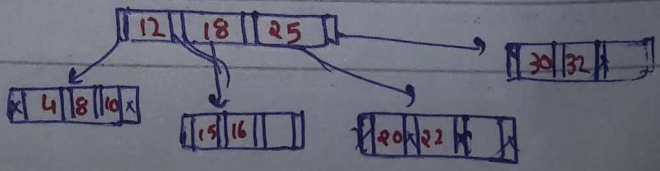
Balanced search Trees

AVL tree | Red Black tree ← Balanced binary search tree →

B Tree | B+ Tree ← Balanced search tree with any degree of node 'p'.

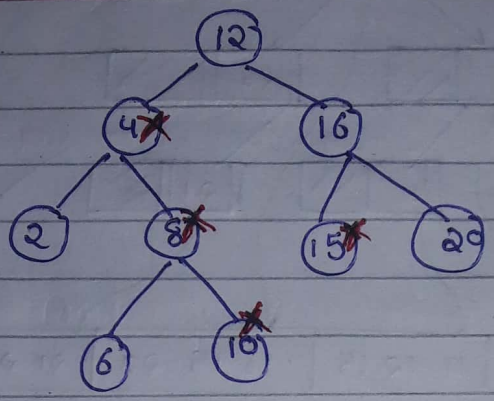
degree : 2





Since there is no restriction of degree in B and B+ tree, ∴ it is preferred for database index.

Note – Lazy deletion :



If we wish to delete 8, we simply set the flag and do not delete it

↓

Later when many are deleted i.e many flags are set

↓

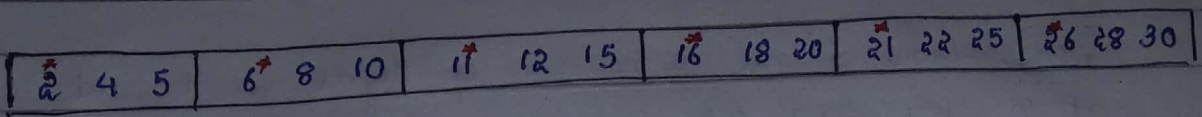
We re-create the tree

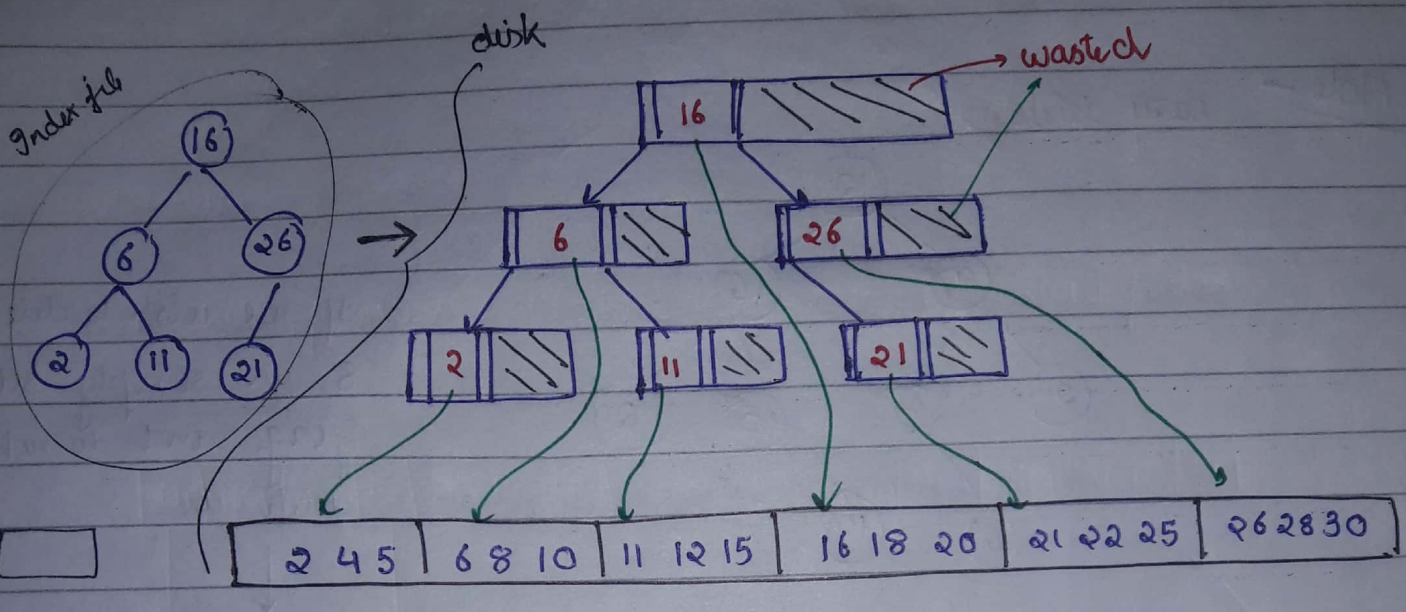
as For n deletion, it takes $\Theta(n \log n)$

**** Ques: Why B/B+ tree used for DB files index rather than BST ??

① Because to reduce access cost
(DB file stored in disk index to DB file must also be in disk and
'Data access from disk is in terms of blocks' (drawback))

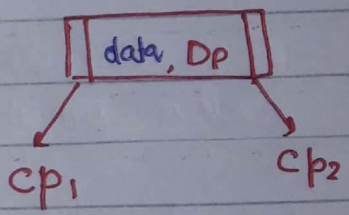
② If AVL tree is used as DB index :





child pointer : points to other tree node (blue line)

data pointer : " " original data file (green line)



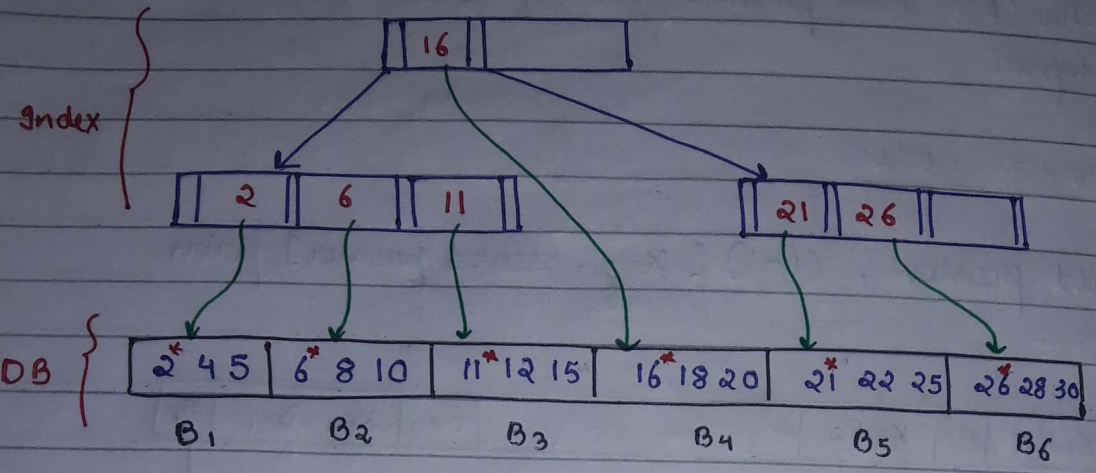
If AVL tree is used for DB index, one entire disk block should allocate for index node and only one key is stored in one node which leads to wastage of disk space allocated for index and more access cost

↓

To avoid this problem

B/B+ Tree is Used.

③ If B/B+ tree used for DB index



- less I/O cost
- less wastage of Disk space.

Conclusion

Main Memory Application

Search over MM data (keys stored in MM)

- less no. of keys, $n = 50$
  - Preferred DS: AVL Tree
- Huge no. of keys, $n = 1,00,000$
  - if AVL tree, then 1,00,000 nodes and 2,00,000 ptr
  - Preferred DS: B Tree

Secondary Memory Application

Search over SM block (data)

- less no. of keys
- Huge no. of keys
  - Preferred DS: B tree / B+ Tree
  - ← I/O cost reduction
  - more i/p →

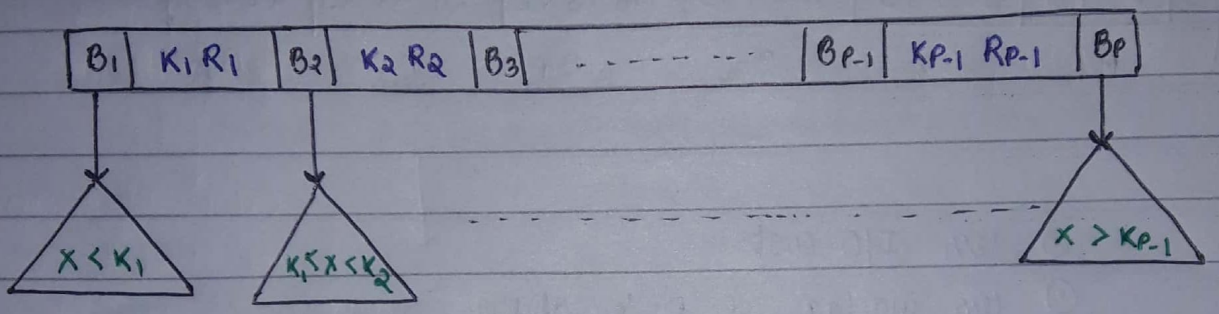
12/11/17

B-Tree —

Order (P): Max possible child pointers in B Tree node (degree)

① Structure of node

P child pointers, (P-1) [Key, record pointer] pairs



- child pointer / block pointer / tree pointer points to the index node.
- Data pointer / Record pointer points to the database block file.

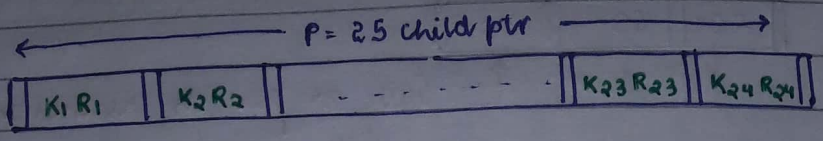
② Every internal node except root with atleast [P/2] child ptr, and ([P/2] - 1) keys and atmost P child ptr and (P-1) keys must store.

③ Root can be atleast 2 child pointers with 1 key and atmost P child ptr with (P-1) keys should be stored.

④ Every leaf node must be at same levels (and) keys in node should be in sequential order

$K_1 < K_2 \text{ ------ } < K_{P-1}$

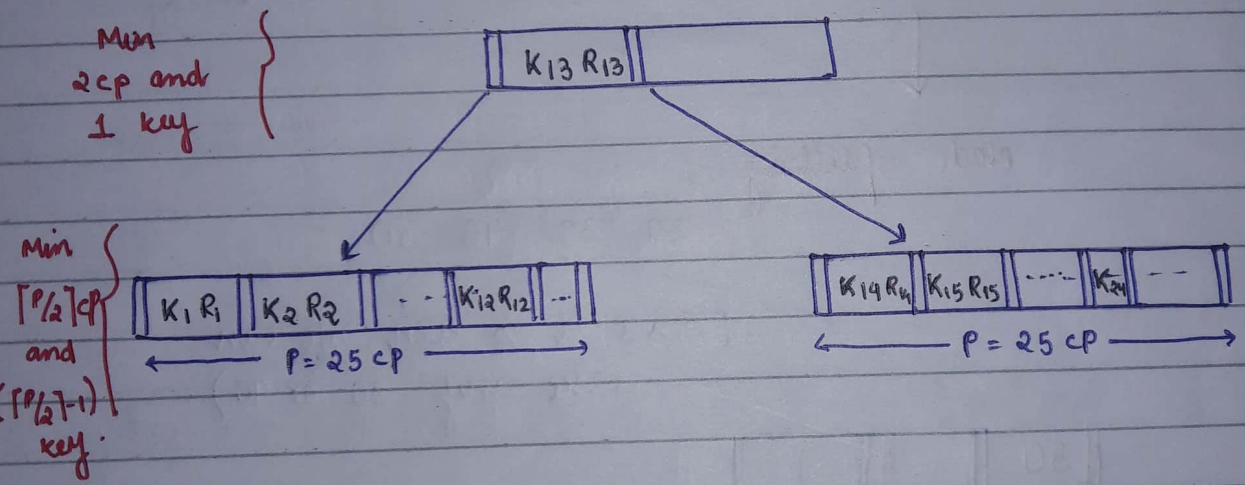
eg - Assume the child pointer P = 25.



Assume, we need to insert (K_{25}, R_{25})
↓
we cannot do as it is full
∴ split.

Node split
(in order to insert new node)

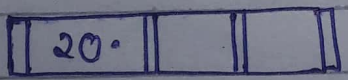
$K_1 < K_2 < \ldots < K_{13} < \ldots K_{25}$



Ques: Construct B Tree with order P: 4 [max cp per node] with following sequence of keys

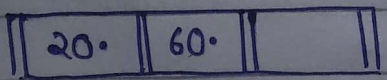
20, 60, 30, 40, 90, 50, 10, 15, 25, 70, 80

Insert 20 -

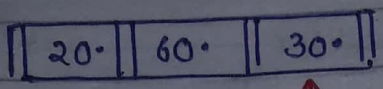


P = 4
∴ (P-1) = 3 keys

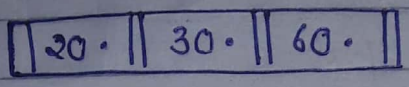
Insert 60 -



Insert 30 –



but as per its restriction (it must be a sequence data)

∴ 

Insert 40 –

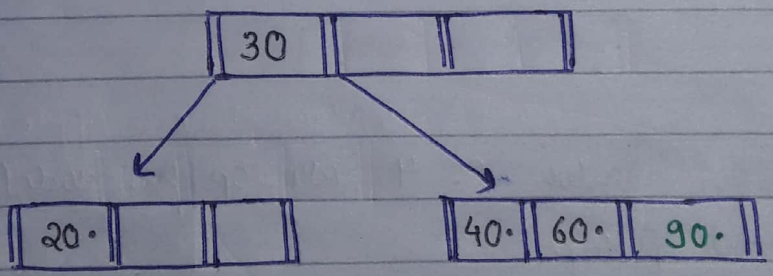
At this level, the node is full ∴ Node splitting.

↓

Node splitting

20 (30) (40) 60

(Since it is even, we can take any 30 or 40)

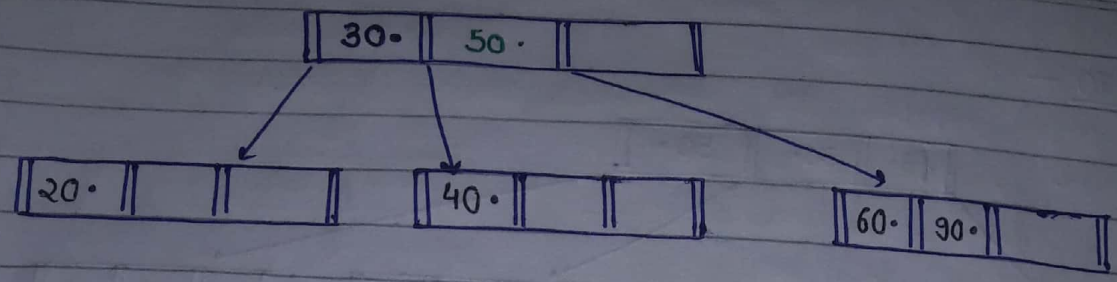


Insert 50 –

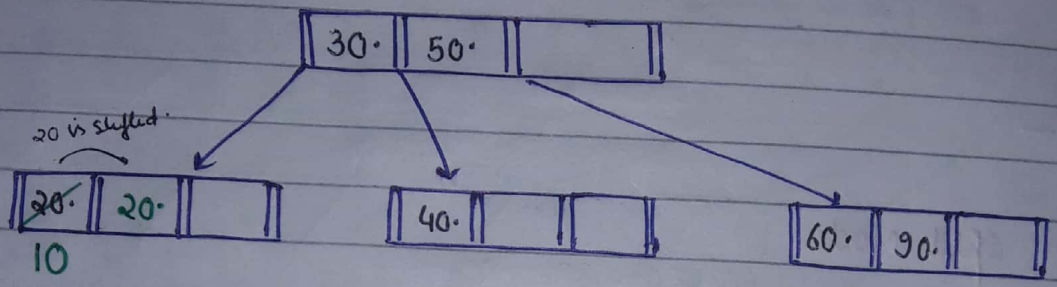
Now to insert 50 ... node is full

∴ Split 40, (50), 60, 90

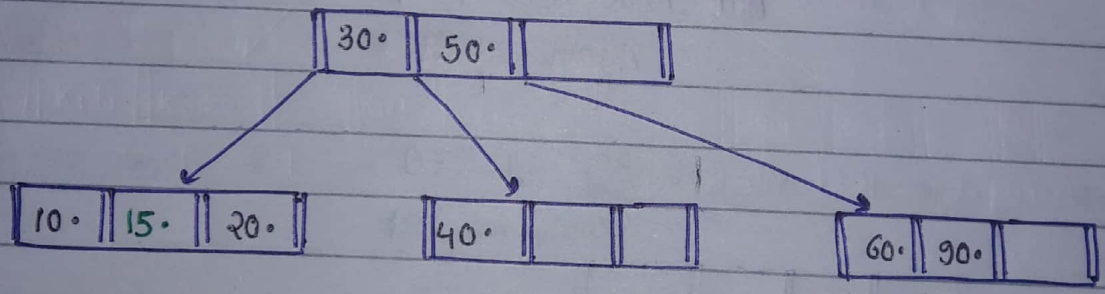
↓ moved to root



Insert 10 —



Insert 15 —

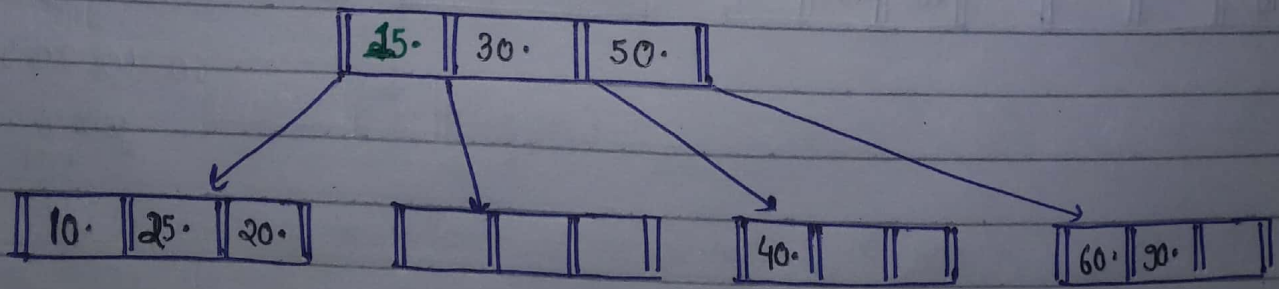


Insert 25 —

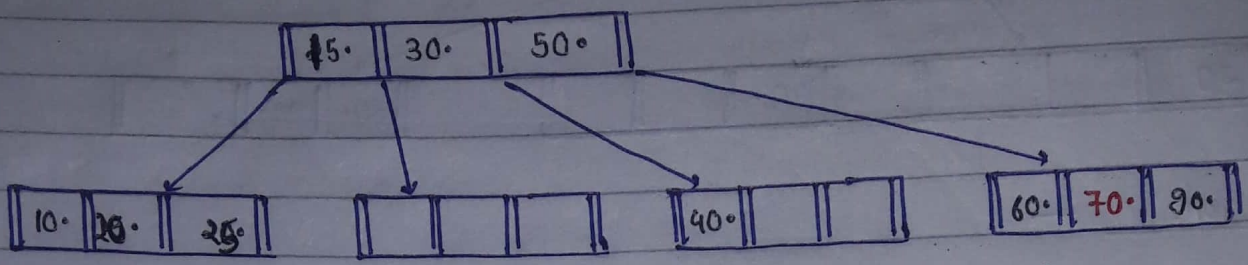
split node

goes to root

10, 15, 20, 25



Insert 70 —



Insert 80 —

Split Node

60 (70) 80 90

goes to root

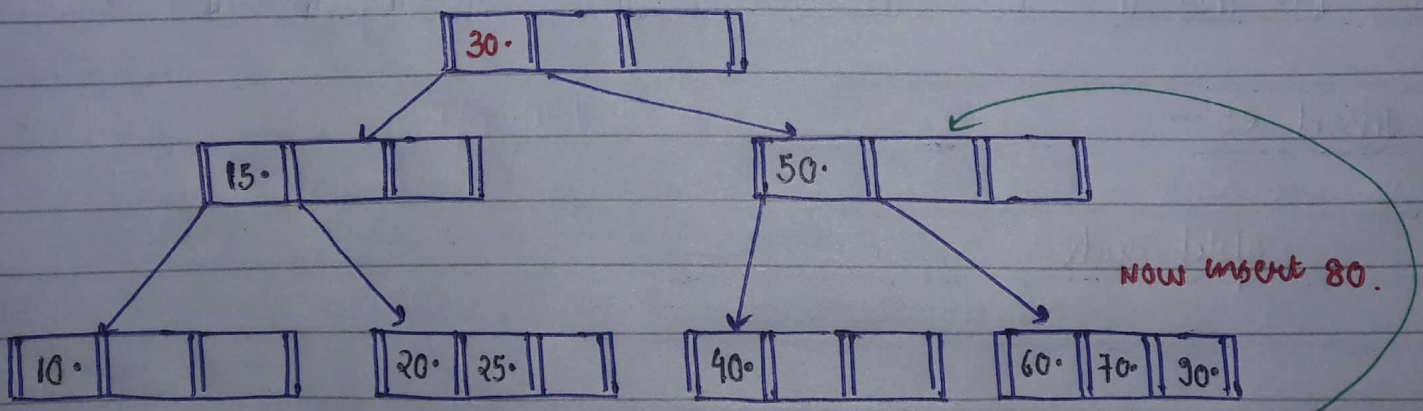
↓

But root also full

∴ Again split

15 (30) 50 70

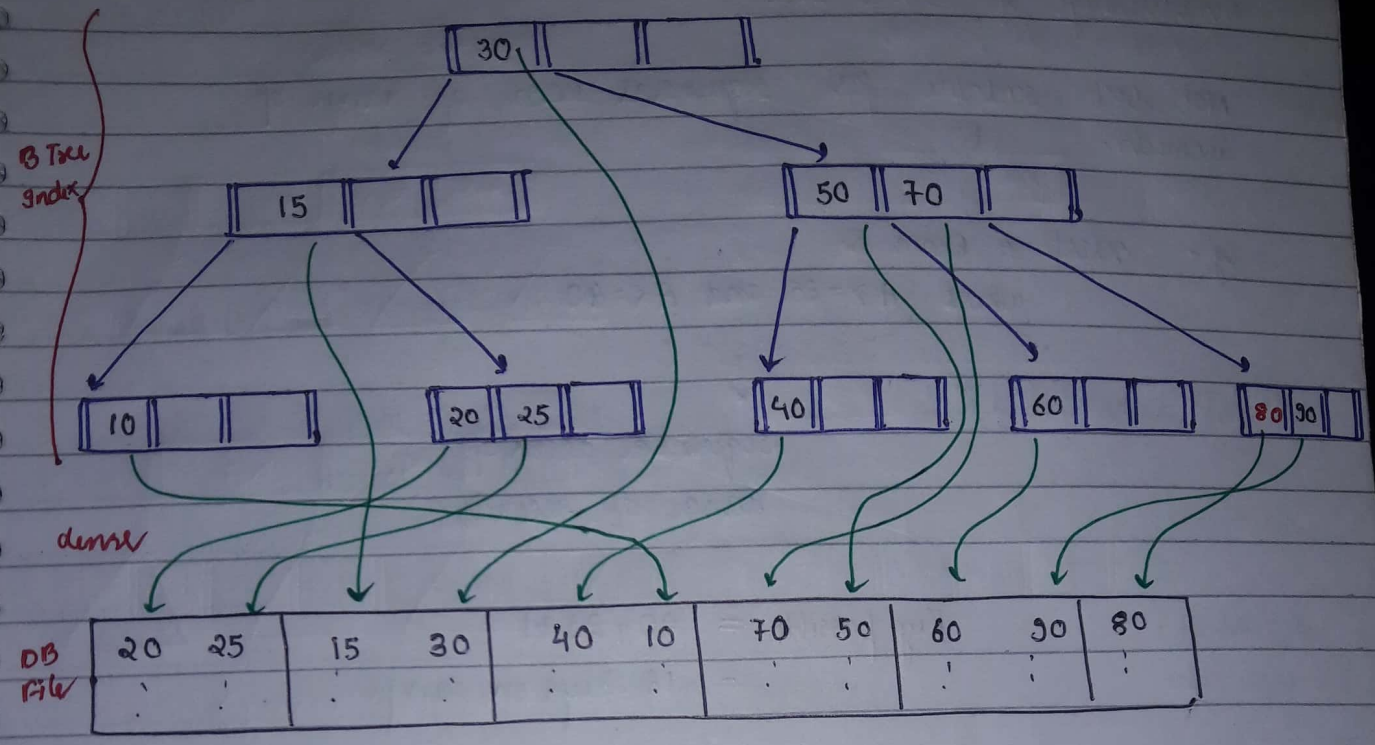
becomes new root



Now insert 80.

60, (70) 80, 90

goes to root



Advantage of B-Tree Index —

B-Tree index is best suitable for random access of any one record.

Eg - select *
From R
where (A=25) ← Any one record access of file.

I/O cost : (K+1) block access

$K \cong O(\log_p n)$

I/O cost of random access query : $O(\log_p n)$

Disadvantage of B Tree index

Not best suitable for sequential access of range of records.

Eg - select * from R
where A >= 25 and A <= 40 ;

↓

sequential access of range of records.

Key possible = 40 - 25 + 1
= 16 keys in range.

∴ Total access cost = 16 * (access cost)
= 16 * $\log_p n$ (large)

cost of successful = K + 1 Block
cost of unsuccessful = K.

∴ I/O cost = $\theta(x \cdot \log_p n)$

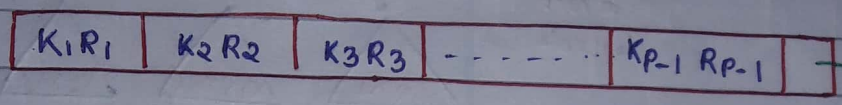
B+ Tree —

Order (P): Max possible pointers in B+ tree node.
(degree)

① Structure of node :

a) Leaf Node —

Consist only (Key, R_P) pairs and one block pointer points to next leaf node.

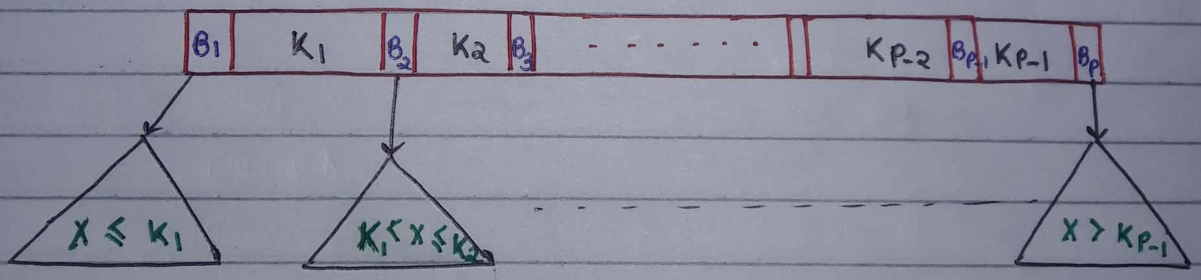


P : No. of pointers.

(pointing to next leaf node)

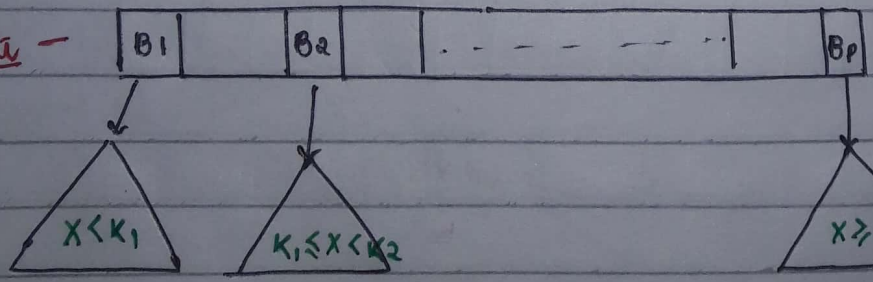
b) Internal Node —

It consist only keys and child pointer and no record pointer.

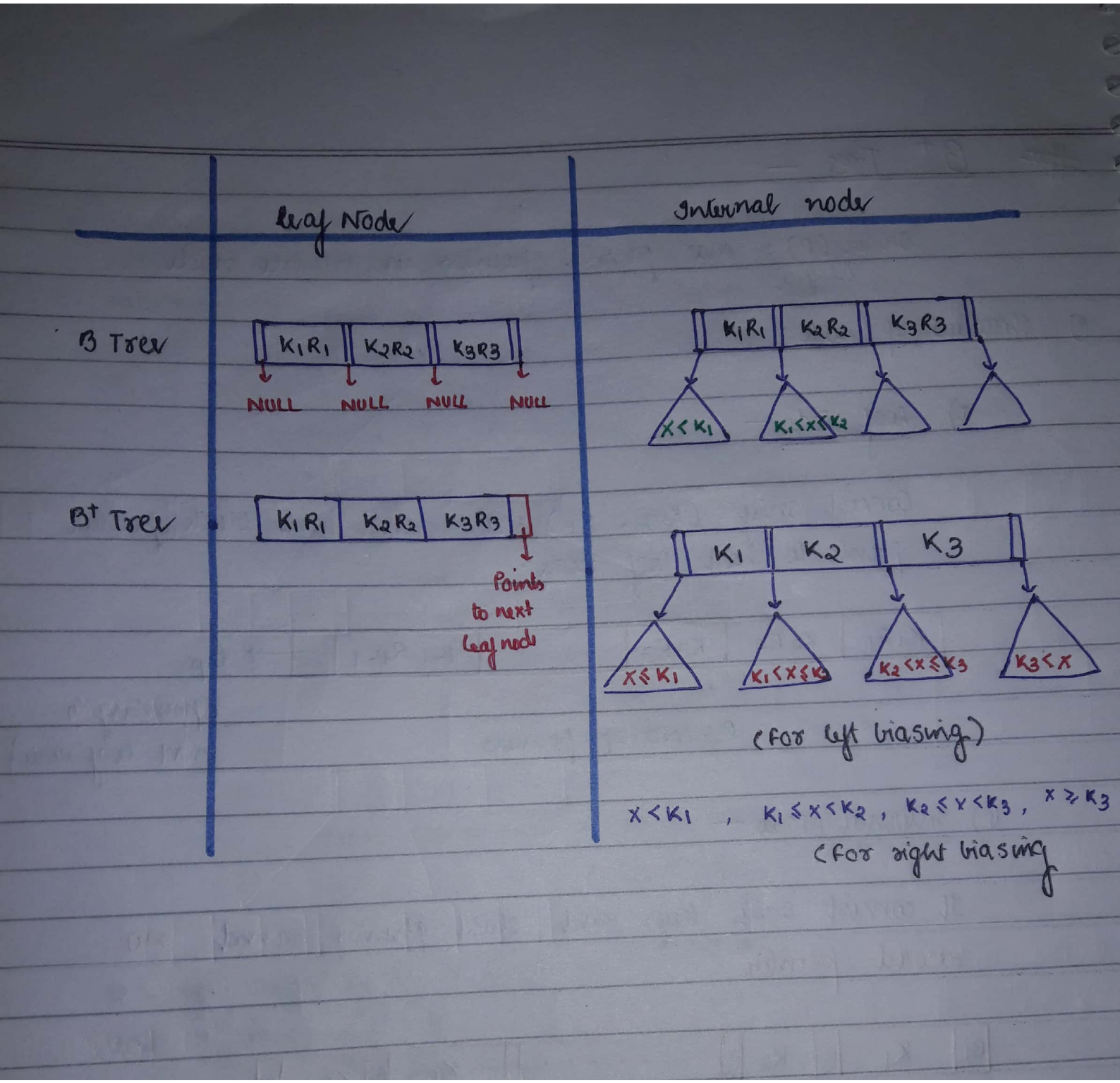


↑ Left biasing
(parent key is maintain towards its left pointer)

Note —



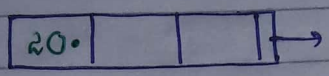
↑ Right Biasing
(Parent maintained towards right pointer)



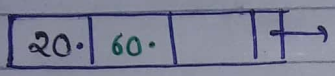
Ques: Construct B^+ tree with order $P = 4$ (max possible pointer per node) with following sequence of keys –

20, 60, 30, 40, 90, 50, 10, 15, 25, 70, 80.

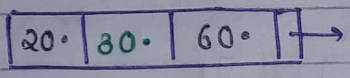
Insert 20 –



Insert 60 –



Insert 30 –

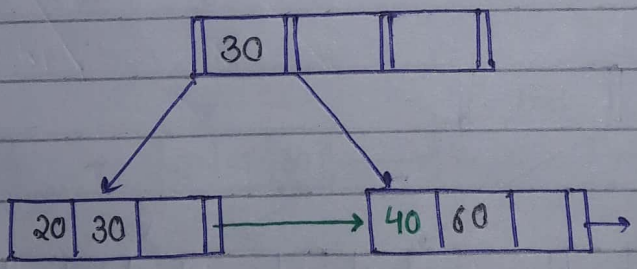


Insert 40 –

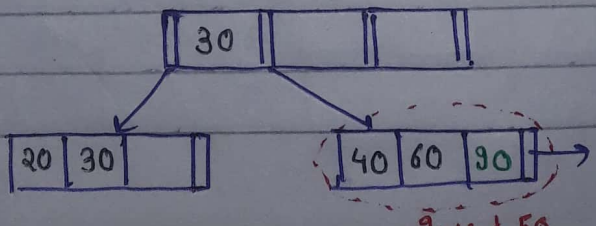
leaf node splitting

20 | 30 | 40, 60

left biasing.



Insert 90 –

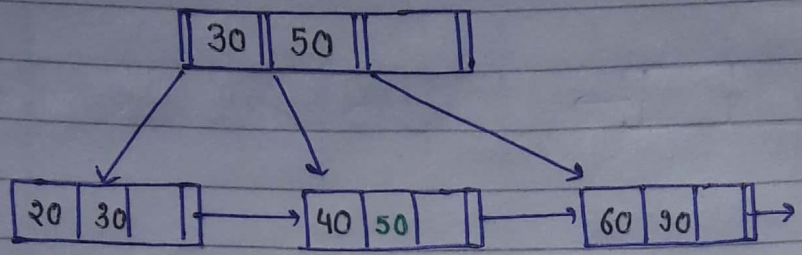


Insert 50

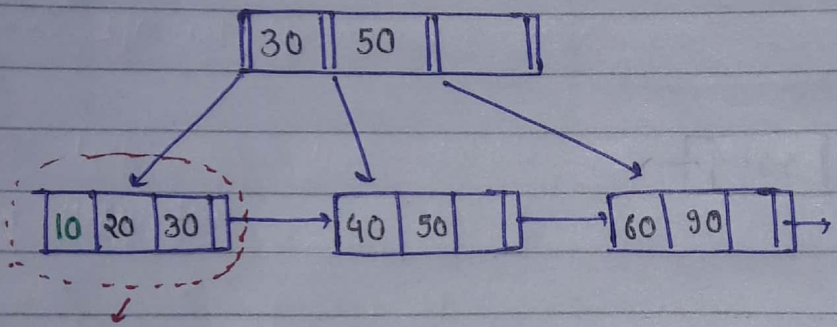
Insert 50 —

split the leaf node

40, 50, 60, 90



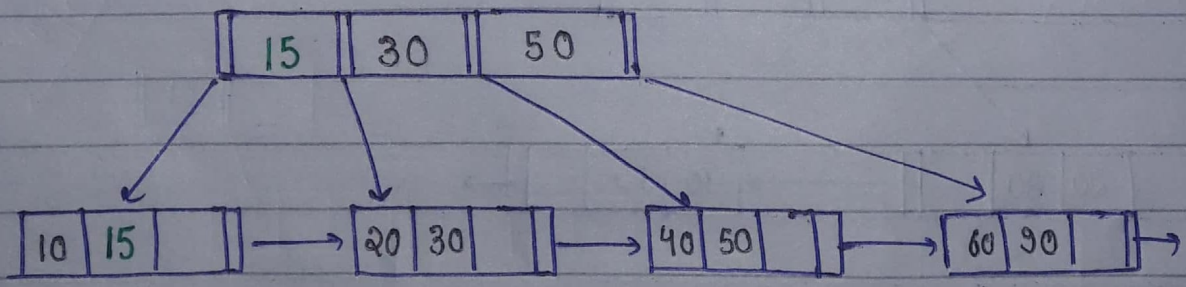
Insert 10 —



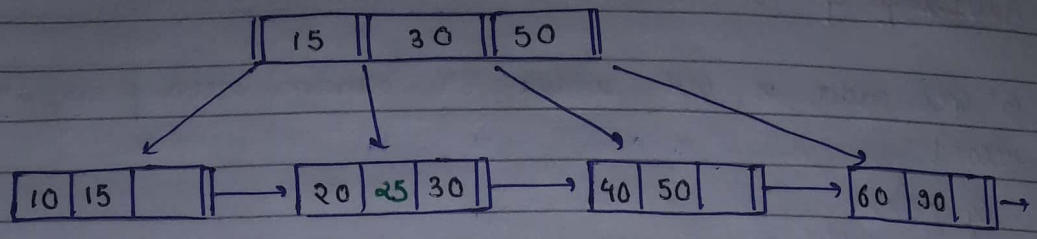
Insert 15 —

split leaf node

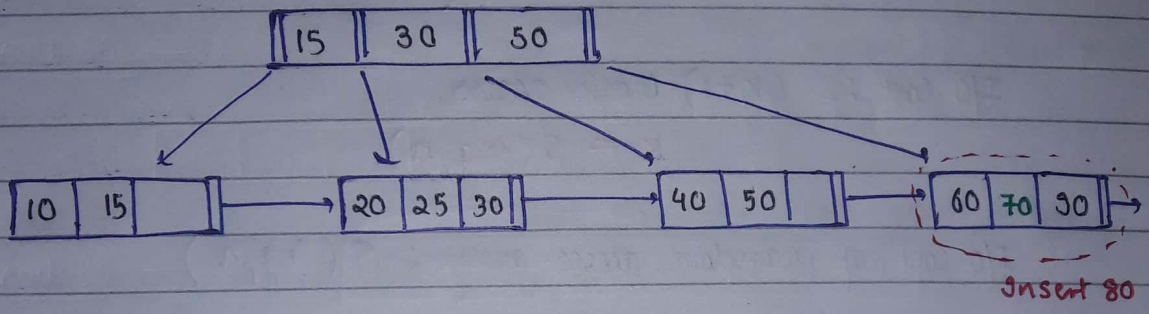
10, 15, 20, 30



Insert 25 –



Insert 70 –



Insert 80 –

Split the leaf node

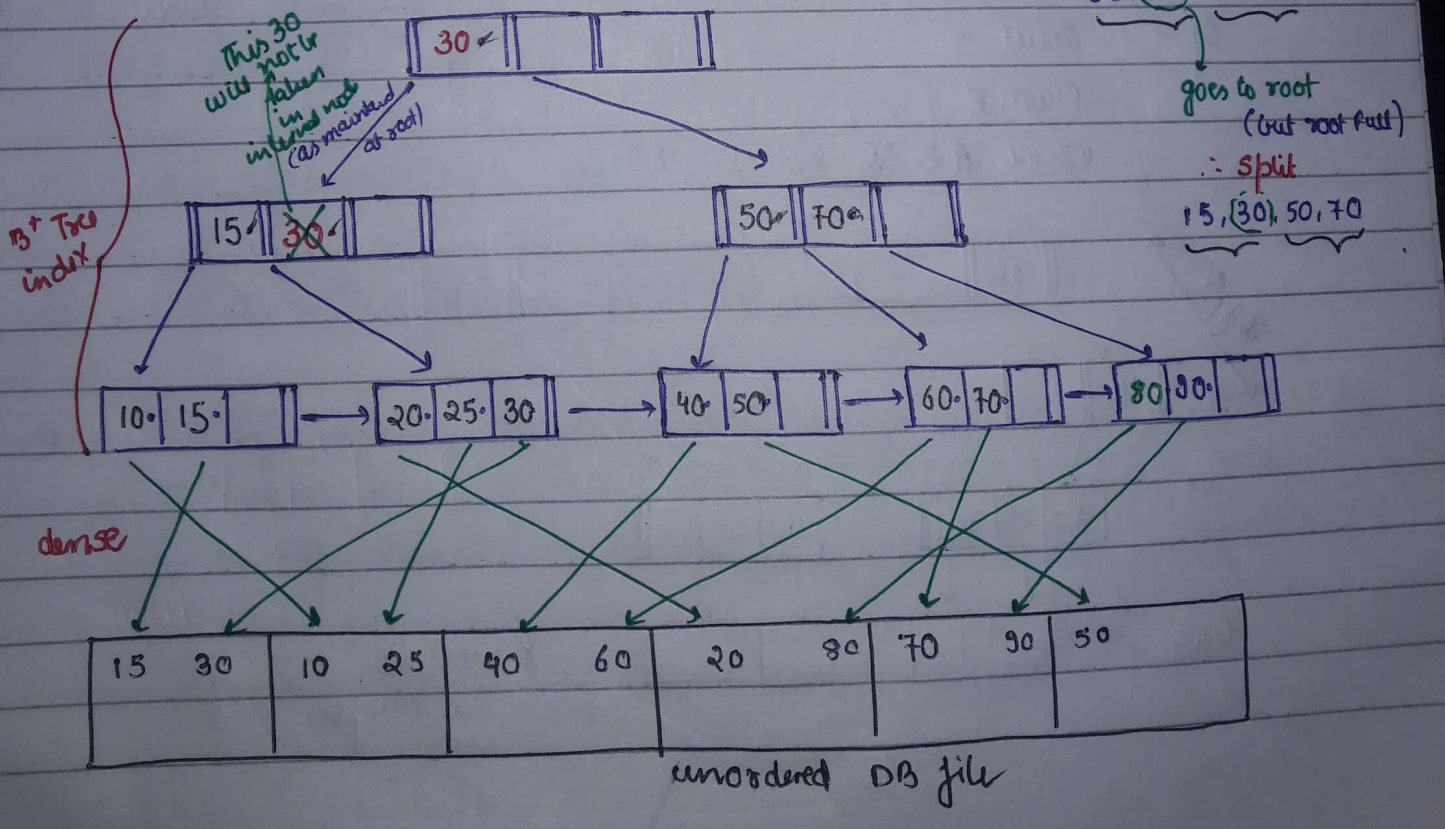
60, (70) 80, 90

goes to root

(but root full)

∴ Split

15, (30), 50, 70



→ Advantage of B+ Tree index —

① B+ tree index is best suitable for random access of any one record.

Eg —

```
Select *
From R
Where (A = 25);
```

→ any one record access of file.

I/O Cost : (K+1) block access

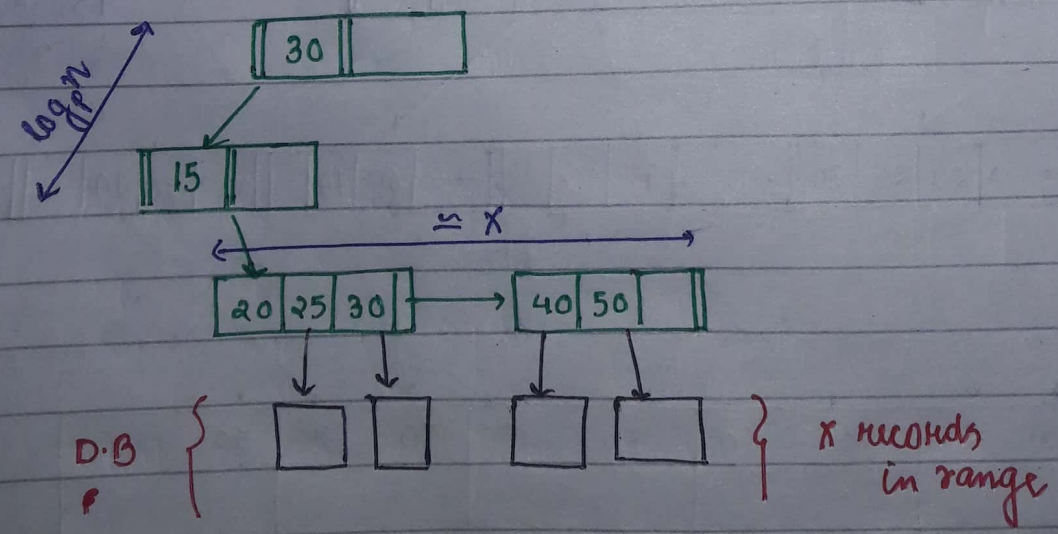
$K = O(\log_p n)$

∴ I/O cost of random access query : $O(\log_p n)$

② B+ tree index best suitable for sequential access of range of records.

Eg —

```
Select *
From R
Where A >= 25 && A <= 40.
```



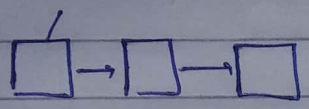
∴ Total access cost = $\log_p n + x + x$

$\simeq \theta(x + \log_p n)$

for successful search.

* For unsuccessful search, there is no cost.

Note - If the file is ordered, then index is not required

 it sequentially accessed.

① In B^+ tree, all keys that are required for indexing must be stored at leaf level.

② Every leaf node, pointed to next leaf node.

③ Leaf node consist (key, data pointer) pairs and one block pointer to point next leaf node.

④ In Internal node consist only keys and child pointer.

⑤ Entire tree is fully left biased (or) right biased.

⑥ In left biasing B^+ tree, keys stored in internal nodes are maximum key of each leaf node except last leaf node.
(No. of internal node keys are equal to No. of leaf node − 1)

*** **Ques:** Find best possible order of B Tree node / B^+ Tree node.

(1) Block size = 1024 Bytes
Search key = 9 Bytes
Block pointer = 6 Bytes
Record ptr = 8 bytes.
what is best possible order of B Tree node if Order P : max possible block ptr in B Tree node ?

Soln:

Order P : Max Bp per node

Block (B Tree node)

$$P * Bp + (P-1)(Keys + Rp)$$

$$P * Bp + (P-1)[Key + Rp] \leq \text{Block size}$$

$$\Rightarrow P * 6 + (P-1)(9+8) \leq 1024.$$

$$\Rightarrow 23P \leq 1041$$

$$\therefore P = \left[\frac{1040}{23}\right]$$

$$= 45 \leftarrow \text{max child pointer per node (Order)}$$

Ans

(2) Block Size = 512 Bytes
$Bp = Rp = 10$ Bytes
Search key = 15 Bytes

what is the best possible order of B tree node if Order P is max possible keys per node.

Soln:

order P : Max possible keys / node

Block (B Tree)

$$(P+1)\,B_P + P(\text{keys} + R_P)$$

$\therefore$ $(P+1)\,B_P + P(\text{key} + R_P) \leq$ block size.

$\Rightarrow (P+1)10 + P(15+10) \leq 512$

$\Rightarrow 35P \leq 502$

$\therefore P = \left\lfloor \frac{502}{35} \right\rfloor$

$= 14$ ← Max possible keys per node

Ans

③ Block size = 2048

$B_P = R_P = 20$ Bytes

Search key = 40 Bytes

What is best possible order of B tree node, if ~~order P:~~

order P: b/w 1 to 2P keys in root

b/w P to 2P keys in other nodes ?

↑ Space allocation is based on maximum capacity.

Order P : Maximum 2P keys per node.

(we compute order w.r.t P)

$$\underbrace{(2P+1)B_P + 2P(key + R_P)}_{\text{Block (B Tree node)}}$$

$\Rightarrow (2P+1)B_P + 2P(key + R_P) \leq$ Block size

$\Rightarrow (2P+1)20 + 2P(40 + 20) \leq 2048$

$\Rightarrow 160P \leq 2048$

40P
80P
40
160P

20

$\therefore P = \left\lfloor \frac{2028}{160} \right\rfloor$

$= 12$ ← order

Ans

④ Block size = 1024 B
search key = 9 B
Block ptr = 6 B
Record ptr = 8 B.
what is the best possible order of ⑤ B+ tree leaf node and ⑥ B+ tree internal node if
Order P: Max possible pointers in B+ tree.

Soln: Internal Node –

Order P: No. of Block pointers.

$$\underbrace{P * B_P + (P-1)\,keys}_{\text{Block (B+ Tree node)}}$$

$\therefore \quad P * Bp + (P-1)\,key \leq$ Block size

$\Rightarrow \quad P * 6 + (P-1)\,9 \leq 1024$

$\Rightarrow \quad 15P \leq 1033$

$$P = \left\lfloor \frac{1033}{15} \right\rfloor$$

$$= 68$$ Ans

Leaf Node –

P : No. of pointers

1 Bp (P-1) Rp

$$(P-1)(Key.Rp) + 1 * Bp$$

$\therefore \quad (P-1)(Key + Rp) + 1 * Bp \leq$ Block size

$(P-1)(9+8) + 6 \leq 1024$

$17P = 1035$

$$P = \left\lfloor \frac{1035}{17} \right\rfloor$$

$= 60$

59 key in leaf node

1 key for Block ponter.

Note —

If size of Block pointer, B_p = size of record pointer R_p, then,

order of B^+ tree leaf node = B^+ tree internal node.

⑤ Block size = 512 B

$B_p = R_p = 10$ B

Search key = 15 B

What is the max possible order of B^+ tree node if order P is max possible keys per node.

Soln:

order P : Max possible keys per node.

Here we can compute any i.e order of internal node or leaf node.

Order of internal node :

$$(P+1) * B_p + (P) \text{ keys}$$

$$\therefore (P+1) * B_p + (P) \text{ keys} \leq \text{Block size}$$

$$(P+1) * 10 + (P) 15 \leq 512$$

$$10P + 15P + 10 \leq 512$$

$$25P \leq 502$$

$$P = \left\lfloor \frac{502}{25} \right\rfloor$$

$$= 20$$

OR

Order of leaf node :

$$P(\text{key} + R_p) + 10P$$

$$\therefore P = 20$$

⑥ Block size = 2048 B

$B_P = R_P = 20$ B ⟶ we can compute order for any node

Search key = 40 B

what is the best possible order of B+ tree node if

order P : b/w 1 to 2P keys in root

b/w P to 2P keys in other node.

↑ max limit taken

leaf / internal node :

$$1 B_P + 2P(\text{keys} + R_P)$$

$\therefore$ $1 B_P + 2P(\text{keys} + R_P) \leq$ block size

$20 + 2P(40 + 20) \leq 2048$

$20 + 80P + 40P \leq 2048$

$120P \leq 2028$

$$P = \left\lfloor \frac{2028}{120} \right\rfloor$$

(OR) $= 16$ Ans

internal node :

$$(2P+1) B_P + 2P * \text{keys}$$

$\Rightarrow$ $P = 16$ Ans

Note- If same size blocks are allocated for B Tree node and B+ node, then

order of B+ tree node > order of B tree node.

[No. of keys stored in B+ tree node] > [No. of keys stored in B tree node]

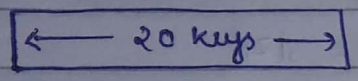
For secondary Memory App.

→ If same size blocks are allocated for B Tree index and B+ tree index, then

[No. of B Tree index nodes for n distinct keys] ≥ [No. of B+ tree index nodes for n distinct keys]

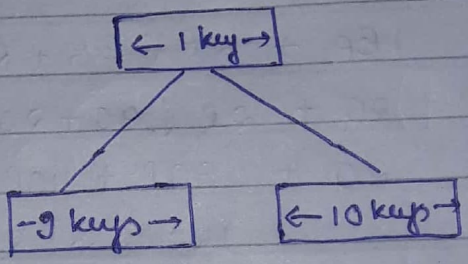
Eg - For n = 20

In B+ Tree



(1 node)

In B Tree



(3 node)

→ [No. of levels of B Tree index for n distinct keys] ≥ [No. of levels of B+ tree index for n distinct keys]

I/O cost — I/O cost

B+ tree index preffered for DB table over B tree because I/O cost of range and I/O cost of random access query is less in B+ tree

(B+ also preffered for random access)

For Main Memory Appl.

→ If order of B Tree node = order of B+ tree node, then,

No. of B tree index nodes for n distinct keys	≤	No. of B+ tree index node for n distinct keys
No. of B tree index level of for n distinct keys.	≤	No. of B+ tree index levels for n distinct keys.

Eg -

B Tree order (P : 14 keys)	=	B+ tree order (P : 14 keys
15 Bp + 14 (Keys + Rp)		14 (Rp + keys) + Bp (or) 14 keys + 15 Bp
512 B		360 B

Both are different

↓

Not possible in secondary memory

↓

∴ Used for MM application as any Bytes can be allocated in it.

Min / Max keys in BTree / B+ tree index for given height (h) –

① Min possible keys in B / B+ tree of order P (max Cp per level) and height h [0, 1, 2, ... h]

Assume, For root : Min 2 Bp and 1 key

other node : Min $\lceil P/2 \rceil$ Bp and $(\lceil P/2 \rceil - 1)$ key

Height		Min Node	Min Bp	Min key
(Root)	0	1	2	1
	1	2	$2\lceil P/2 \rceil$	$2(\lceil P/2 \rceil - 1)$
	2	$2\lceil P/2 \rceil$	$2\lceil P/2 \rceil^2$	$2\lceil P/2 \rceil (\lceil P/2 \rceil - 1)$
	3	$2\lceil P/2 \rceil^2$	$2\lceil P/2 \rceil^3$	$2\lceil P/2 \rceil^2 (\lceil P/2 \rceil - 1)$
	⋮	⋮	⋮	⋮
	h	$2\lceil P/2 \rceil^{h-1}$	~~$2\lceil P/2 \rceil^h$~~	$2\lceil P/2 \rceil^{h-1} (\lceil P/2 \rceil - 1)$

+

Min possible keys in B Tree with order P and height h :

$$\Rightarrow 1 + 2(\lceil P/2 \rceil - 1) + 2\lceil P/2 \rceil (\lceil P/2 \rceil - 1) + 2\lceil P/2 \rceil^2 (\lceil P/2 \rceil - 1) \cdots$$

$$\Rightarrow 1 + 2(\lceil P/2 \rceil - 1) \underbrace{\left[1 + \lceil P/2 \rceil + \lceil P/2 \rceil^2 + \cdots \lceil P/2 \rceil^{h-1} \right]}_{\text{GP series}}$$

$\Rightarrow 1 + 2\left(\lceil p/2 \rceil - 1\right)\left[\frac{\lceil p/2 \rceil^h - 1}{\lceil p/2 \rceil - 1}\right]$

$\Rightarrow 1 + 2\left(\lceil p/2 \rceil^h - 1\right)$

$\Rightarrow \boxed{2\left\lceil \frac{p}{2} \right\rceil^h - 1} \simeq \Theta\left(\lceil p/2 \rceil^h\right)$

↑ Min keys for B tree.

Also,

Min Nodes $= \Theta\left[\left(\frac{p}{2}\right)^h\right]$

Min possible keys in B+ tree with order p and height (h) —

Only nodes at lay level is counted as all other are repeated value

↓

$$\boxed{2\left\lceil \frac{p}{2} \right\rceil^{h-1}\left(\left\lceil \frac{p}{2} \right\rceil - 1\right)} \simeq \Theta\left(\left\lceil \frac{p}{2} \right\rceil^h\right)$$

(2) Max possible keys in B/B+ Tree with order p and height h —

Assume, Max p Bp and (p-1) keys per node.

↓

Height	Max node	Max BP	Max keys
0	1	P	$P-1$
1	P	$P*P$	$P*(P-1)$
2	P^2	P^2*P	$P^2(P-1)$
3	P^3	P^3*P	$P^3(P-1)$
⋮	⋮	⋮	⋮
h	P^h		$P^h(P-1)$

$$\text{Max node} = \left(\frac{P^{h+1}-1}{P-1}\right)$$

$$\text{Max key} = \left(P^{h+1}-1\right)$$

Max keys in B+ Tree with order P and height (h) $= P^{h+1}-1 \simeq \Theta(P^h)$

Also,

Max key in B+ tree with order P and height (h) $= P^{h+1}-1$

Height of B tree with order p and n distinct key :-

a) Min Height

(Max possible keys per node)

$$n = p^{h+1} - 1$$

$$\boxed{h = \log_p(n+1) - 1} \simeq O(\log_p n)$$

b) Max Height

(Min possible key per node)

$$n = 2\left\lceil \frac{p}{2} \right\rceil^h - 1$$

$$\frac{n+1}{2} = \left\lceil \frac{p}{2} \right\rceil^h$$

$$\boxed{h = \log_{\lceil p/2 \rceil}\left(\frac{n+1}{2}\right)} \simeq \log_{p/2} n/2 \simeq O(\log_p n)$$

Note :-

Keys of B/B+ tree : $O(\lceil p/2 \rceil^h)$ ..... to ..... $O(p^h)$

Height of B tree : $\log_p n$ to $\log_{p/2} n/2$

B+ tree : $\log_p n$ to $\log_{p/2} n/2$

Ques: Order P : b/w 2 to P block ptr in root node
b/w $\lceil P/2 \rceil$ to P block ptr in other internal node
b/w $\lceil P/2 \rceil - 1$ to $(P-1)$ keys in leaf node
of B Tree / B+ tree.

① what is the min keys and node in B Tree / B+ tree of order P: 5 and level: 4 ?

② what is the max keys and nodes in B Tree / B+ tree of order P: 5 and level 4.

} design from root to leaf

Soln:

① Min Keys / Nodes of level 4 and order P:5

level	Min node	Min Bp	Min key.
1	1	2	1
2	2	$2 * \lceil P/2 \rceil$ $\Rightarrow 2 * \lceil 5/2 \rceil \Rightarrow 2*3$	2 * 2
3	6	6 * 3	6 * 2
4	18	–	18 * 2
	27		

Min key in B tree = 53

Min key in B+ tree = 36 (only last level i.e leaf node 18*2)

Min nodes in B / B+ tree = 27

② Max keys and nodes.

level	Max nodes	Max BP	Max keys
1	1	5	4
2	5	5 * 5	5 * 4
3	25	25 * 5	25 * 4
4	125	—	125 * 4
	156		

Max key in B tree → 624

Max nodes in B/B+ tree = 156.

Max key in B+ tree = 125 * 4 = 500 (only last i.e leaf.)

Bulk loading B+ Tree —

(Construction of B+ Tree in bottom up Approach)

- It is the design of B+ tree from the leaf nodes to the root node.

Procedure:

① Sort the keys in ascending order which are used for indexing.

② Design the leaf nodes ~~and~~, distribute the keys into leaf node (No. of leaf node)

③ Design internal node

If 'x' nodes at level (ℓ), then
x child pointers at level (ℓ-1)

Distribute 'x' child pointer into nodes (No. of nodes at (ℓ-1))

Repeat step ③ until root.

Ques: No. of keys for index = 2500
order p = 7 (max ptr per node of B+ tree)

[Assume: 2 to p pointers for root
⌈p/2⌉ to p pointers for other nodes]

a) How many min levels of B+ tree index?
(Min nodes)

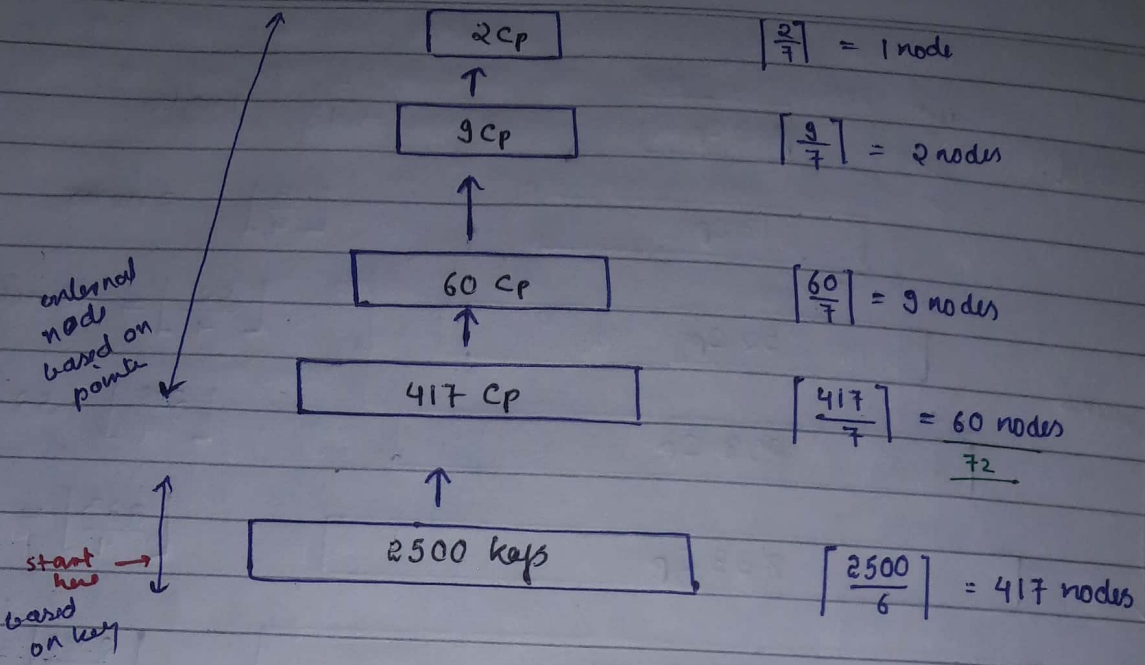
b) How many max levels of B+ tree index?
(max nodes.

// Solve using Bulk loading B+ tree. (when leaf to root)

Soln:

① Min level of B+ Tree index:
(Max node fill factor)

max: p = 7 ptr
key = p-1
= 7-1 ⇒ 6 keys.



∴ Min index level = 5 Ans

Min index blocks = 417 + 72
= 489 nodes. Ans

(2) Max level of B+ tree index

Min fill factor Root : Min = 2 Cp and ~~Max~~ 1 key

Other nodes : Min = ⌈p/2⌉ and ~~Max~~ key = p-1
= ⌈7/2⌉ = 4-1
= 4 Cp => 3

$\left\lfloor \frac{2500}{4} \right\rfloor \Rightarrow 833$

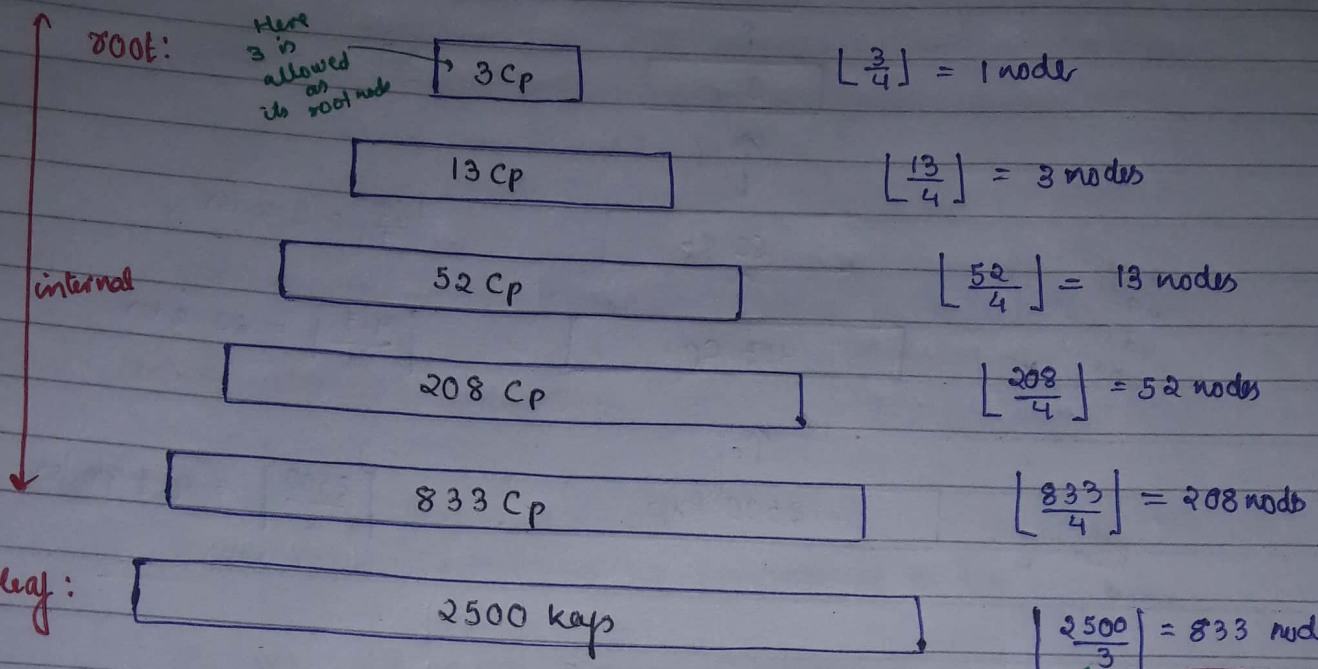
we take floor value
ie
if we take ceil value

$\left\lceil \frac{10}{3} \right\rceil = 4$ node

[3key] [3key] [3key] [1key]

this is not allow

∴ we take ceil value



∴ Total levels of index = 6

⇒ Max level = 6 Ans

Max index blocks = 833 + 277
(833: leaf, 277: internal)

= 1110 Ans

Note

B-Tree Problem —

{ ① Find Min/Max keys of given h levels
② " " " levels of given n keys } Design root to leaf

B+ tree Problem —

① Find min/max key of given h levels
design root to leaf

② Find min/max levels of given n keys ?
Design leaf to root (Bulk loading B+ tree)

Join Algorithms —

① Nested Loop Join Algo
② Block Nested Loop Join Algo

Nested Loop Join Algo :

Let relation R with n tuples occupied in x blocks
relation S with m tuples occupied in y blocks.

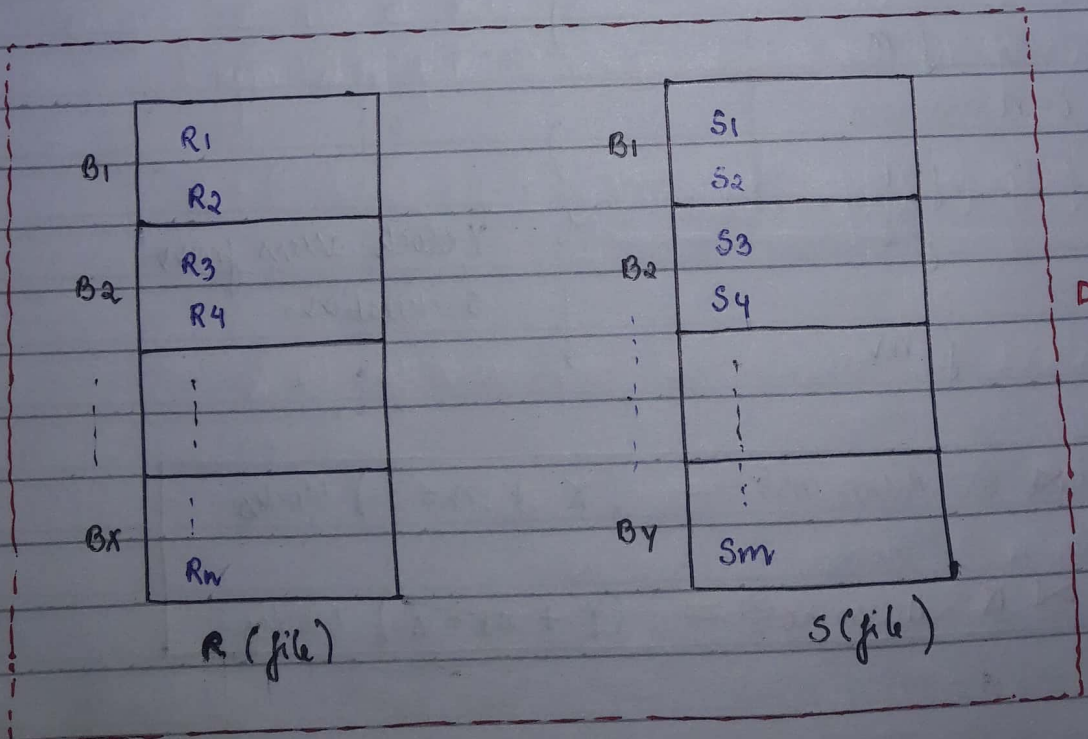
$R \bowtie S \Rightarrow$

```
for (i = 1 ; i ≤ n ; i++)
    for (j = 1 ; j ≤ m ; j++)
        comp (Ri, Rj  Join condition)
```

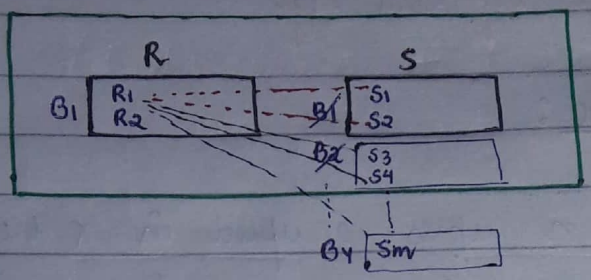
Record no. of R

Record no. of S

drawback — It takes more access cost to join if the main memory space allocated to join is limited.



Case 1 : Assume only 2 blocks of MM allocated for Join (At any time for one block of R data and one block of S data can be stored in MM)



Main memory space for Join

for i = 1

- j = 1
- j = 2
- ⋮
- j = m

} Y block access from S relation

i = 2

- j = 1
- j = 2
- ⋮
- j = m

} Y blocks access from S Relation

⋮

i = n

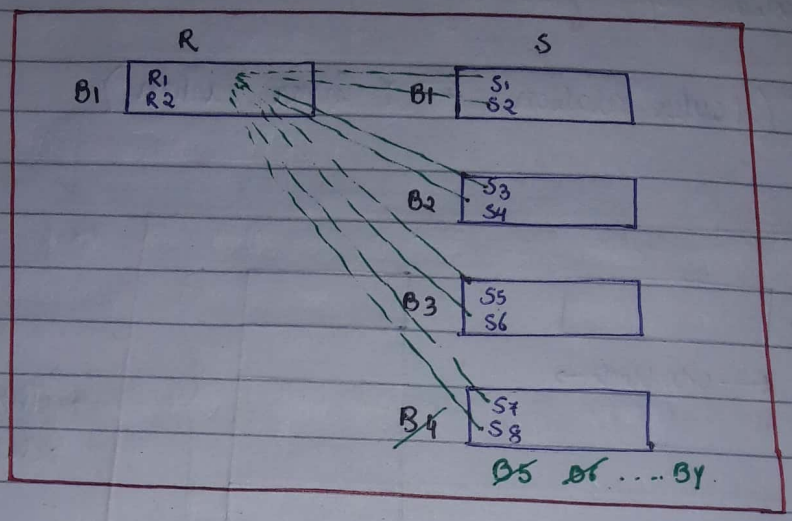
- j = 1
- j = 2
- ⋮
- j = m

} Y block access from S relation.

R ⋈ S	Access cost =	(X + n * Y) blocks
S ⋈ R	Access cost =	(Y + m * X) blocks

Case 2 : Assume 5 blocks of MM allocated for join

a) $R \bowtie S$ Access cost :



for $i = 1$

$j = 1$
$j = 2$
$\vdots$
$j = m$

} Y block of S relation accessed

$i = 2$

$j = 1$
$j = 2$
$\vdots$
$j = m$

} (Y-3) blocks of S relation accessed

$\vdots$

$i = n$

$j = 1$
$\vdots$
$j = m$

} Y-3 blocks of S relation assed

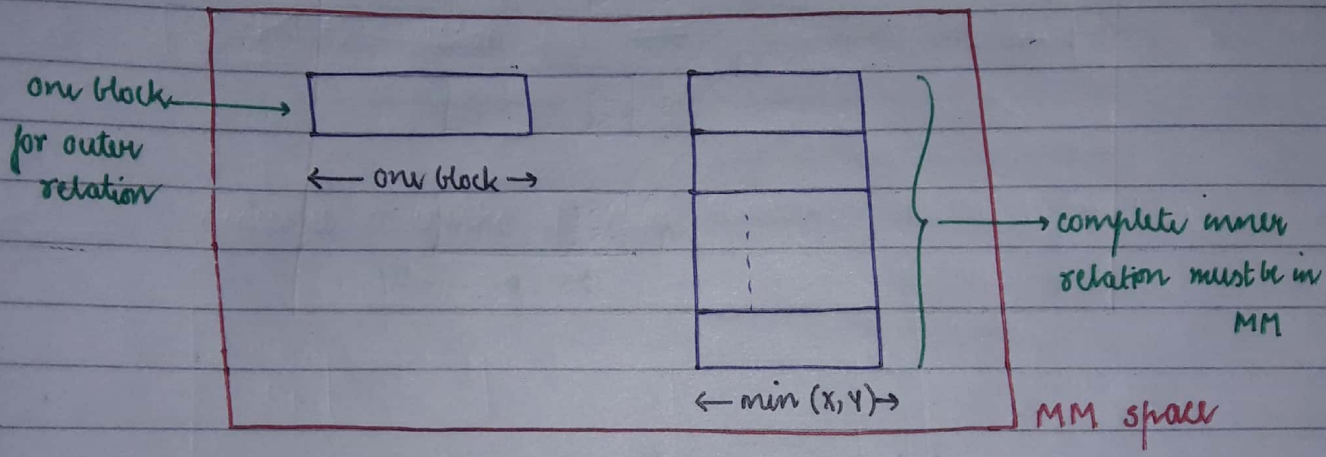
① $R \bowtie S$ Access cost = $X + Y + (m-1)(Y-3)$ blocks

② $S \bowtie R$ Access cost = $Y + X + (m-1)(X-3)$ blocks

Same for both the algo (Nested and Block Nested loop Join)

Case 3: How many min main memory blocks are required to Join R, S without any repeated data access from Disk to main memory using Nested loop join algo.

(1 outer relation + Y inner relation)



Min MM space = Min (X, Y) + 1

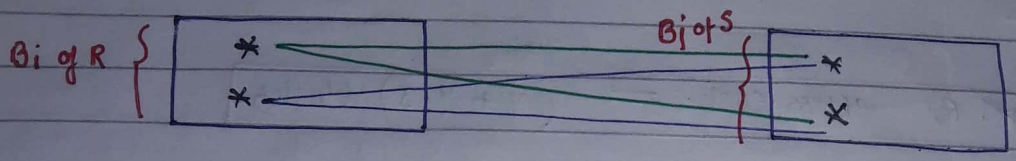
∴ {Min (X, Y) + 1} block of MM is required to avoid repeated data access from DISK to MM to perform Join of R and S using Nested loop Join Algo.

Block Nested Loop Join Algo —

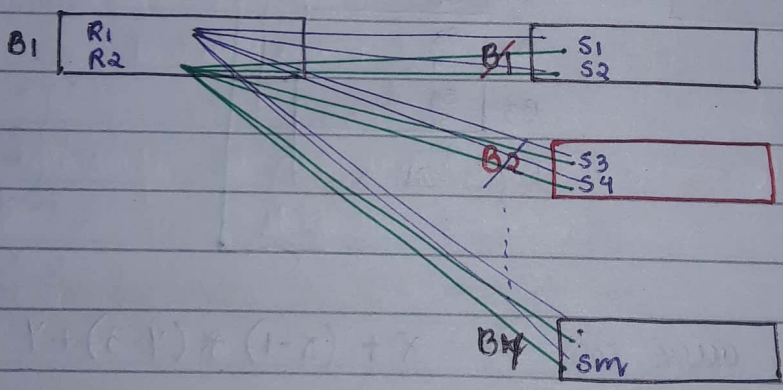
- Here, instead of records, the loop runs for blocks
- Rel R with X blocks of record and
  Rel S with Y blocks of record.

$R \bowtie S \Rightarrow$

```
for (i=1 ; i≤X ; i++)
    for (j=1 ; j≤Y ; j++)
        JOIN (ith block record of R,
              jth block record of S)
```



Case 1: Only 2 blocks of MM allocated for JOIN.



B1 is replace and B2 is in MM

FOR $i = 1$

$j = 1$
$j = 2$
$\vdots$
$j = Y$

} Y blocks of S data accessed

$i = 2$

$j = 1$
$j = 2$
$\vdots$
$j = Y$

} Y blocks accessed from S

$\vdots$

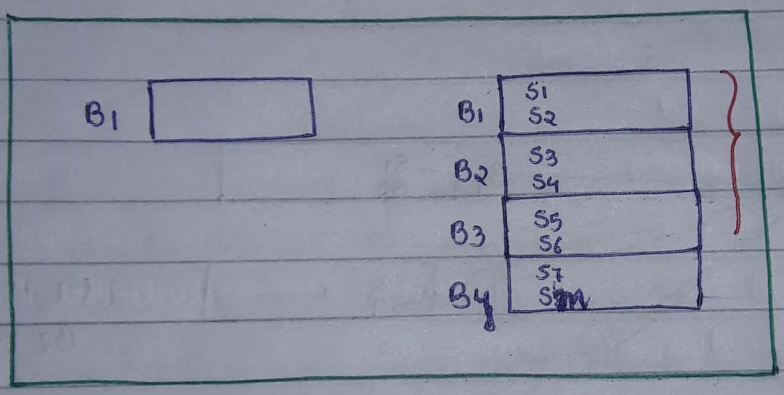
$i = X$ $j = 1$ ... $j = Y$ } Y block accessed from S

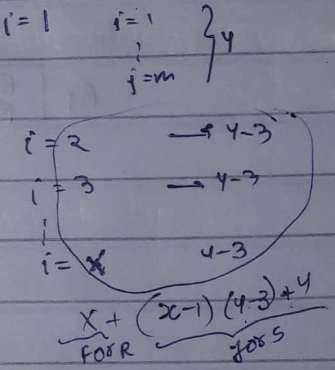
Advantage — Less access cost as compared to Nested loop Join algo, if main memory space is limited.

$R \bowtie S$ access cost $=$ $(X + X \cdot Y)$ blocks

$S \bowtie R$ access cost $=$ $(Y + Y \cdot X)$ blocks

Case 2: Assume 5 blocks of main memory allocated for Join.





$R \bowtie S$ access cost $= X + (X-1) * (Y-3) + Y$

$S \bowtie R$ access cost $= Y + (Y-1)(X-3) + X$

ER Model

- ER diagrams are used to represent high level database designs (diagrametic representation of DB design)

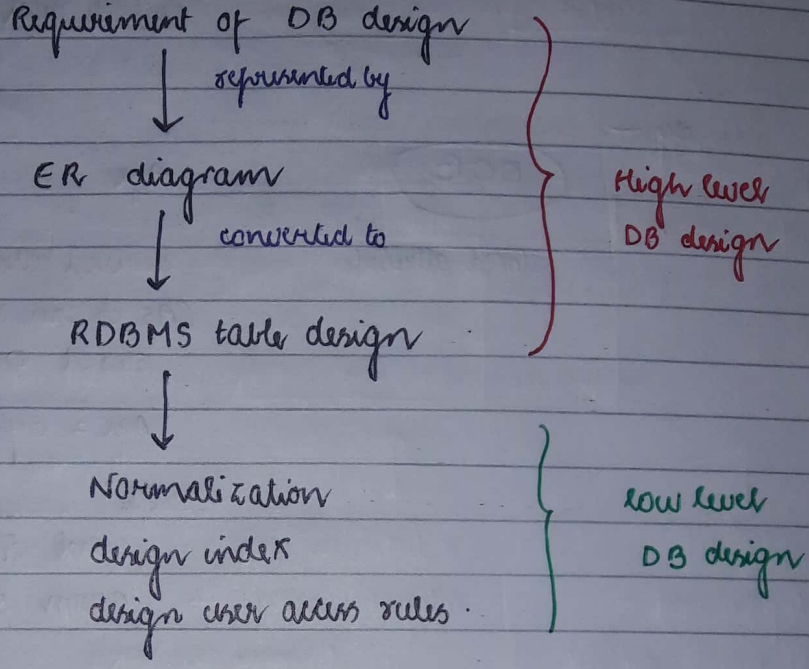
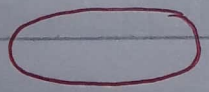


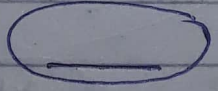
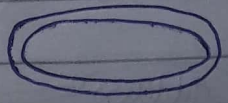
Fig: Database development steps

Main Components in ER model —

1. Attributes
2. Entity sets
3. Relationship sets.

Attributes :

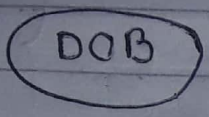


- Key attribute → 
- multivalued attribute → 

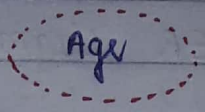
- **derived attribute** ⟶ value of attribute is derived from value of stored attribute

'dotted oval'

Eg –



stored attribute



derived attribute

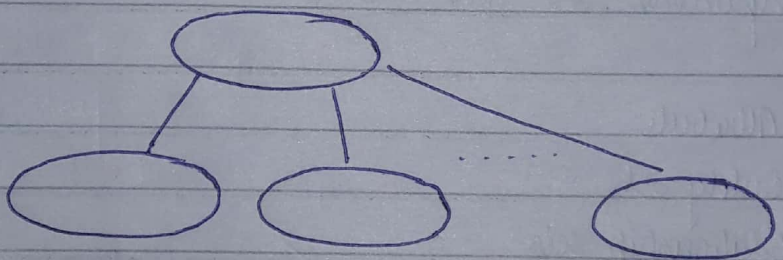
(As it can be derived by stored DOB)

Age = Sysdate – DOB

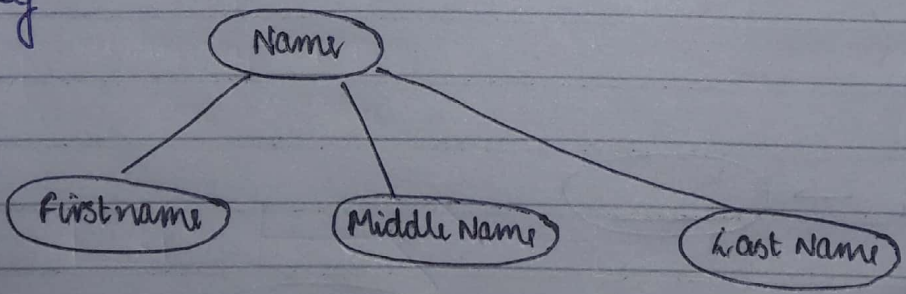
In SQL:

```
Select Sid, (sysdate – DOB) as age
From Student;
```

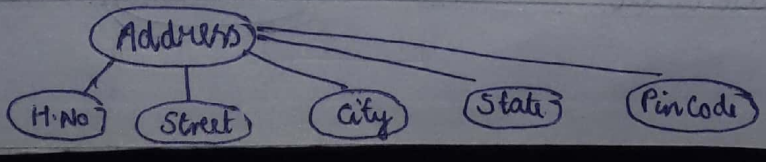
- **Composite Attribute** ⟶ Attribute that can be represented by 2 or more attributes.



Eg –



(or)



Tables are designed based on type of oper^{n} performed -

① If mostly application uses 'full address only'

stud (Sid , address)

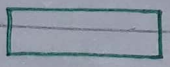
↳ take address as a single field

② If mostly application uses city , state , pincode

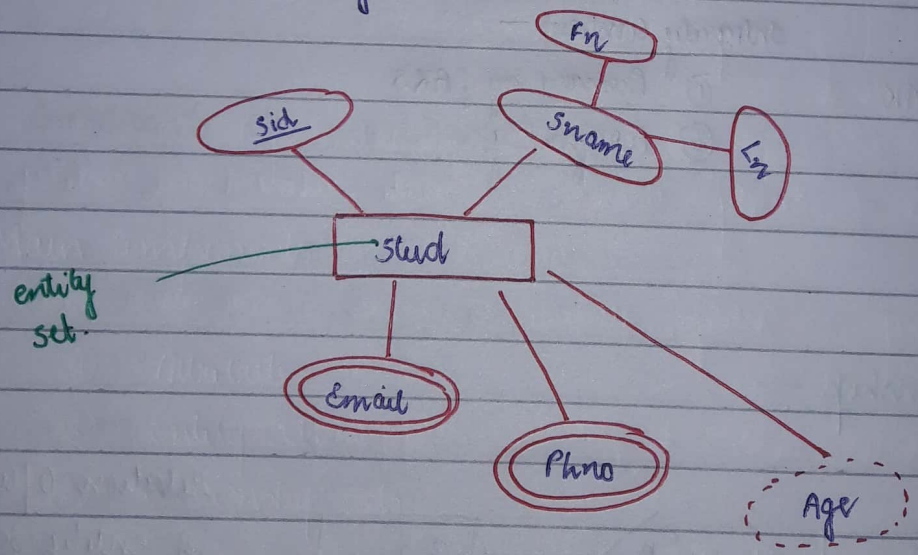
stud (Sid , Hno , Street , City , State , Pin)

together address.

Entity Set —



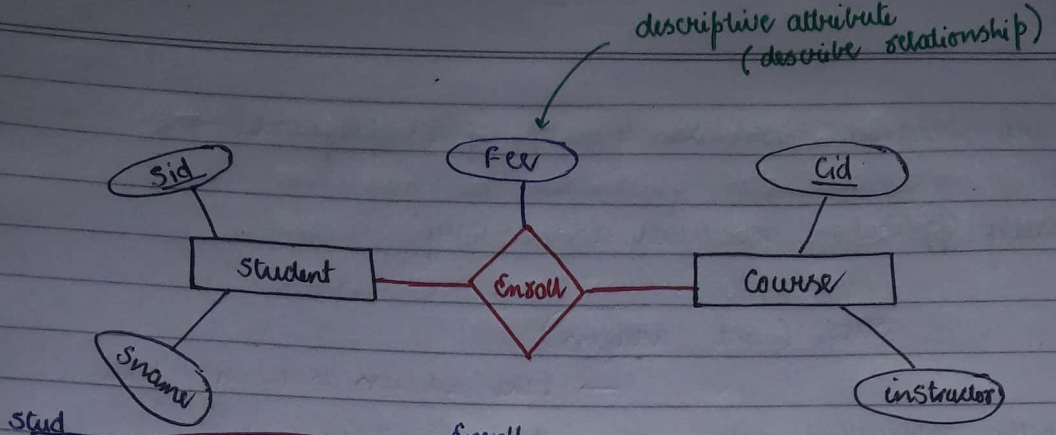
Set of similar entity (object / tuples)



Here, 'Stud' is entity set.

Relationship Set —

Used to relate two or more entity set.



Stud

Sid	Sname

Enroll

Sid	Cid	Fee
S1	C1	5000
S2	C1	4000
S2	C2	5000

Course

Cid	instr

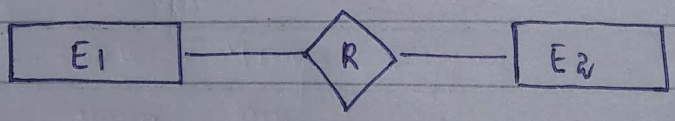
'Fee' attribute is neither purely related to stud nor to course

∴ we represent it on diagram called descriptive attri

Entity set
Integrity const –
① Primary key / AK

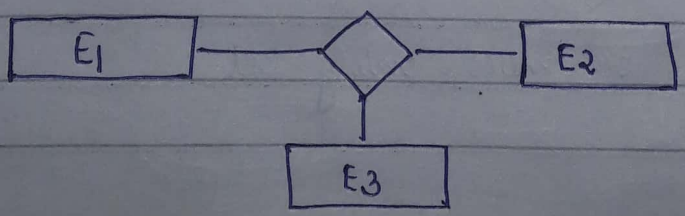
relationship set
Integrity constant –
① Primary key / AK's
② Foreign key.

Binary relationship –



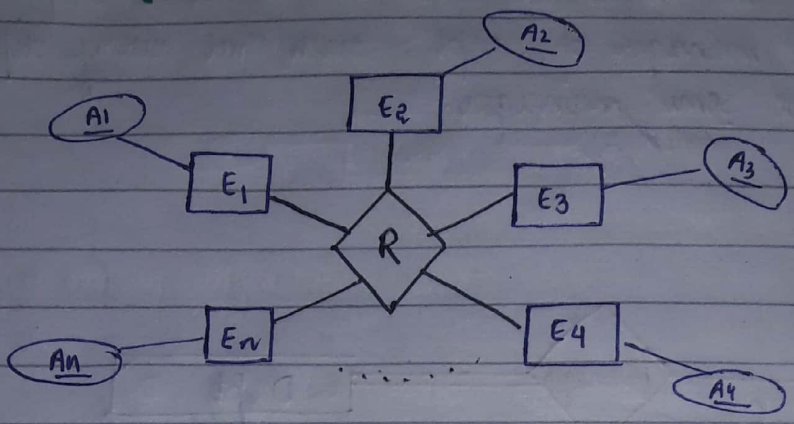
relation b/w
2 entity set.

Ternary relationship set –



relation b/w
3 entity set

n- ary relationship set —



R (A1, A2, ---- An, ... —)

↳ some descriptive attr.

Min attributes needed = n

Foreign key = n

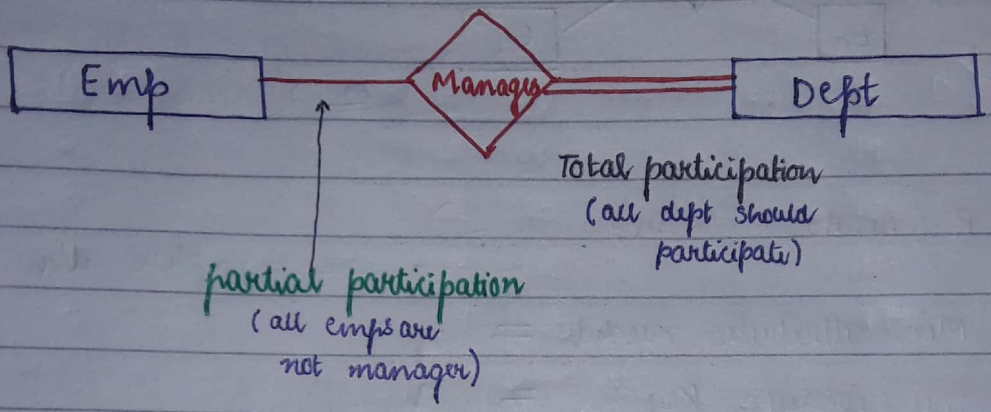
Constraints of relationship set —

1. Mapping (cardinality)
2. Participation

Participation —

- If every entity of entity set is related to relationship set (must cond.), then it is called Total Participation (i.e must be 100% participation) denoted by ═══
- Otherwise, called Partial Participation (≤ 100% participation) denoted by ——

Eg – Emp and Dept are entity sets, & manages is relationship set used to relate emp, who are manager of dept, such that every dept must have some (atleast 1) manager.



Mapping Constraint —

One – Entity allowed to relate atmost one entity (→)

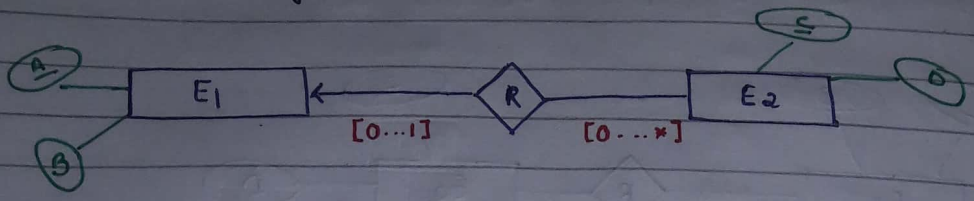
Many – Entity allowed to relate 0 or more entities (—)

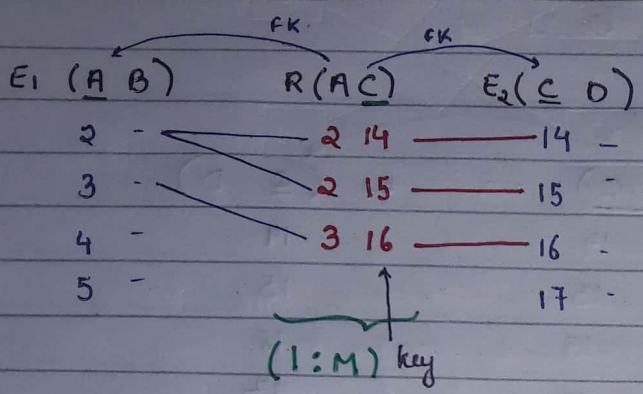
* Binary relationship set possible mapping —

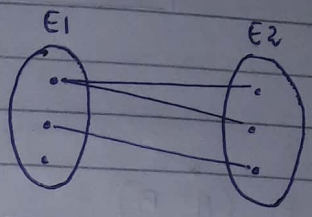
① One : Many mapping
② Many : One "
③ Many : Many "
④ One : One "

} 'Candidate keys' of relationship set depends on mapping constraint

⇒ One : Many mapping relationship set (R) b/w E_1, E_2 entities :







one object of E_1 relate many E_2 (Many)
one " " E_2 " only one E_1 (one)

$E_1 - E_2$ many
$E_2 - E_1$ one

candidate key of rel set R (AC) = {C} ← key of many side

for 1:M mapping

⇒ RDBMS tables design for ERD —

E_1 (A B) R (A C) E_2 (C D)

can be joined.

FK

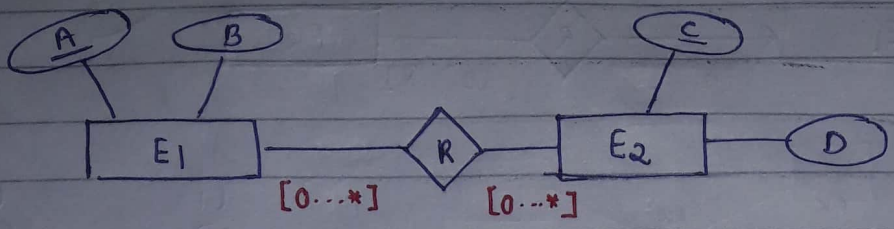
E_1 [A	B]	RE_2 [C	D	A]
2		14		2
3		15		2
4		16		3
5		17		Null

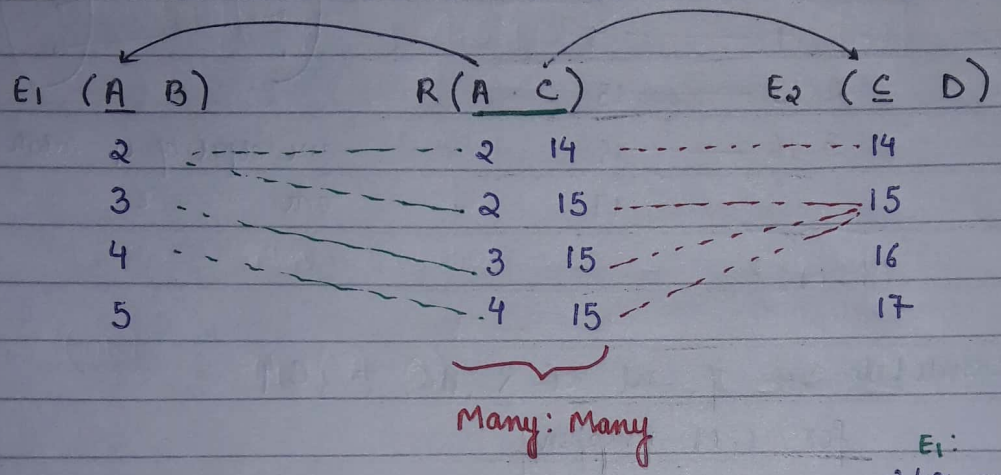
∴ Min 2 tables with 1 foreign key

→ Many : One mapping

Similar to above just E1 — R → E2 cand key = A

→ Many : Many mapping





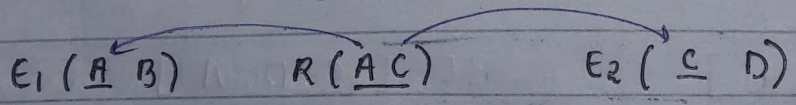
E1:
2 has many mapping
2 → 14, 15 many

E2:
15 has many mapping
15 → 2, 3, 4 many

Candidate key of rel set R(AC) = {AC}
for M:M relation

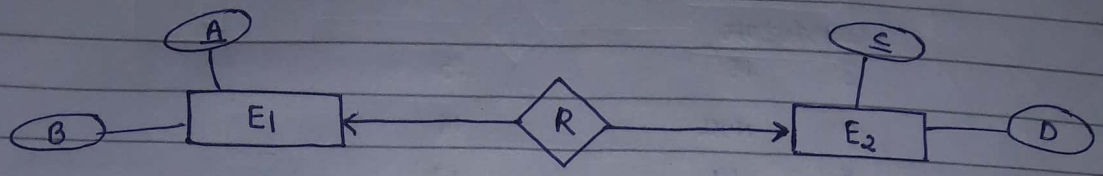
↓

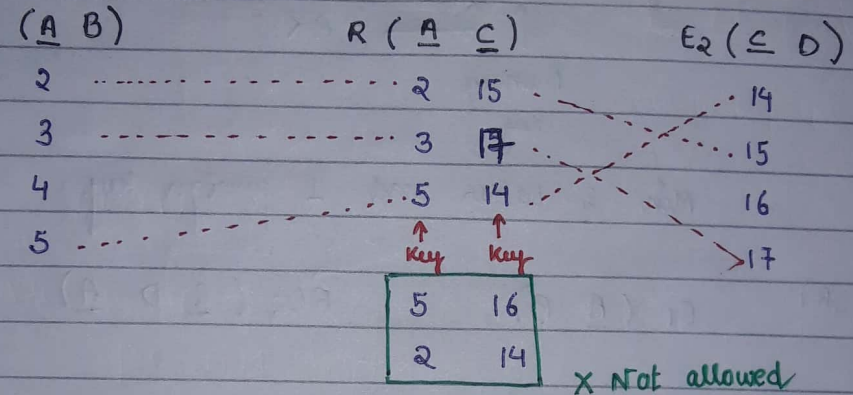
RDBMS design for above ERD



Min 3 relational table and 2 foreign key

→ One : One Mapping relationship b/w E_1 and E_2 with Partial participation at both ends.





∴ candidate keys of relationship set R (A C) is = {A, C}

for 1:1 mapping

↓

RDBMS design for above ER diagram

Even if A and C both are key, still we cannot join all into a single table because of partial participation (as it causes some violation i.e NULL values occur) over left side (E_1 side) and right side (E_2 side)

A	B	C	D
2	-	15	-
3	-	17	-
4	-	Null	NULL
5	-	14	-
NULL	Null	16	-

cand key = {A, C}

X No primary key

(i.e no field whose value does not contain NULL)

∴ FK

E_1 R (A B C) E_2 (C D)

2	15	16	-
~~Candidate~~			
3	17	15	-
4	Null	16	-
5	14	17	-

candidate key = { A, C }

↑ ↑ ← AK.

Primary key

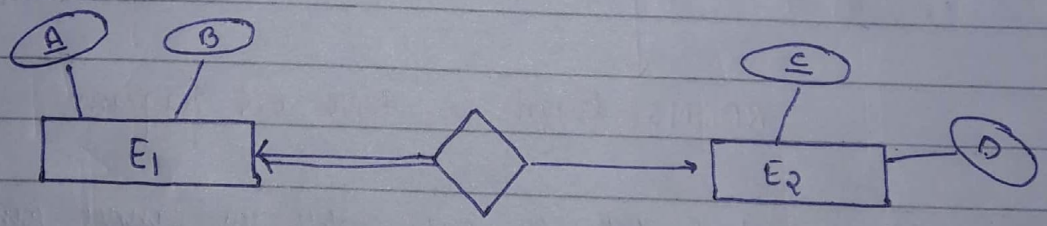
∴ Min 2 tables and 1 foreign key.

(OR)

E_1 (A B) RE_2 (C D A)

→ One: One Mapping relationship b/w E_1 E_2 entity set with atleast one end with total participation.

a)



⊙ E_1 R E_2 (A B C D)

↑ ↑

one relational table — alternate key — Primary key.

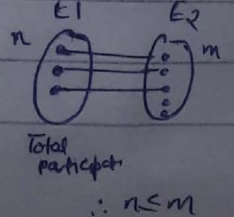
$E_1 \rightarrow C$ ≃ No nulls

↑ total participation

⊙ E_1 E_2 with n & m entities respectively.

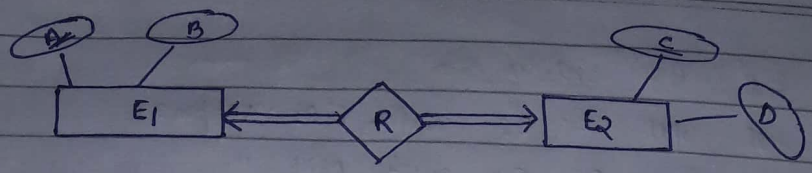
'R' rel with 1:1 and Total participation at E_1

$(n \leq m)$ guaranteed



∴ $n \leq m$

6)



One relational table,

$E_1 R E_2$ (A B C D)

candidate key = { A, C }

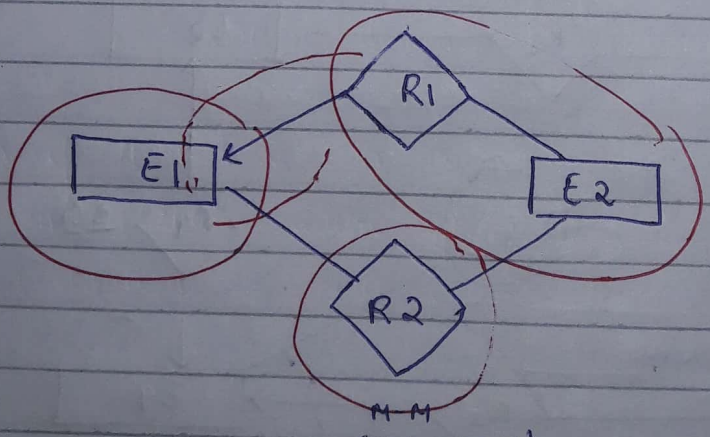
key with NO Nulls

At both side there is total participation.

$\therefore \quad n = m$

13/11/17

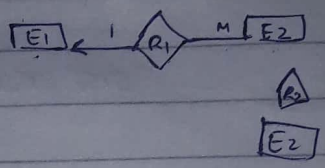
Ques: E_1, E_2 entity sets. R_1, R_2 relationship sets b/w E_1 and E_2 with 1:M and M:N mapping. How many min relational tables required.

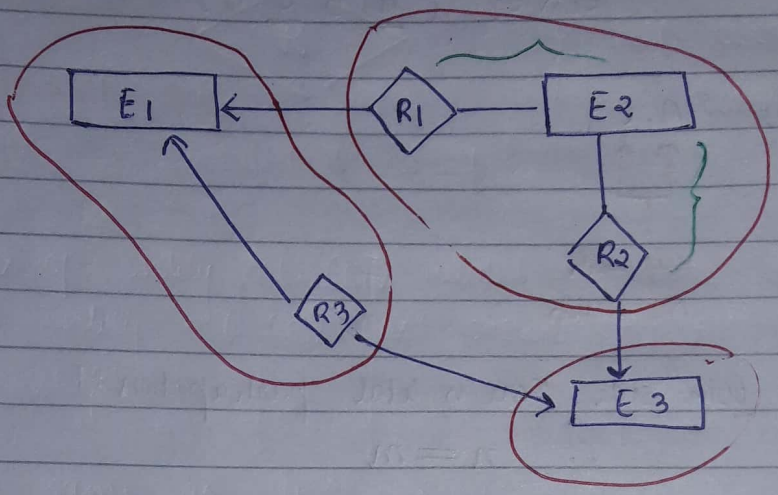


(seperate table)

3 tables

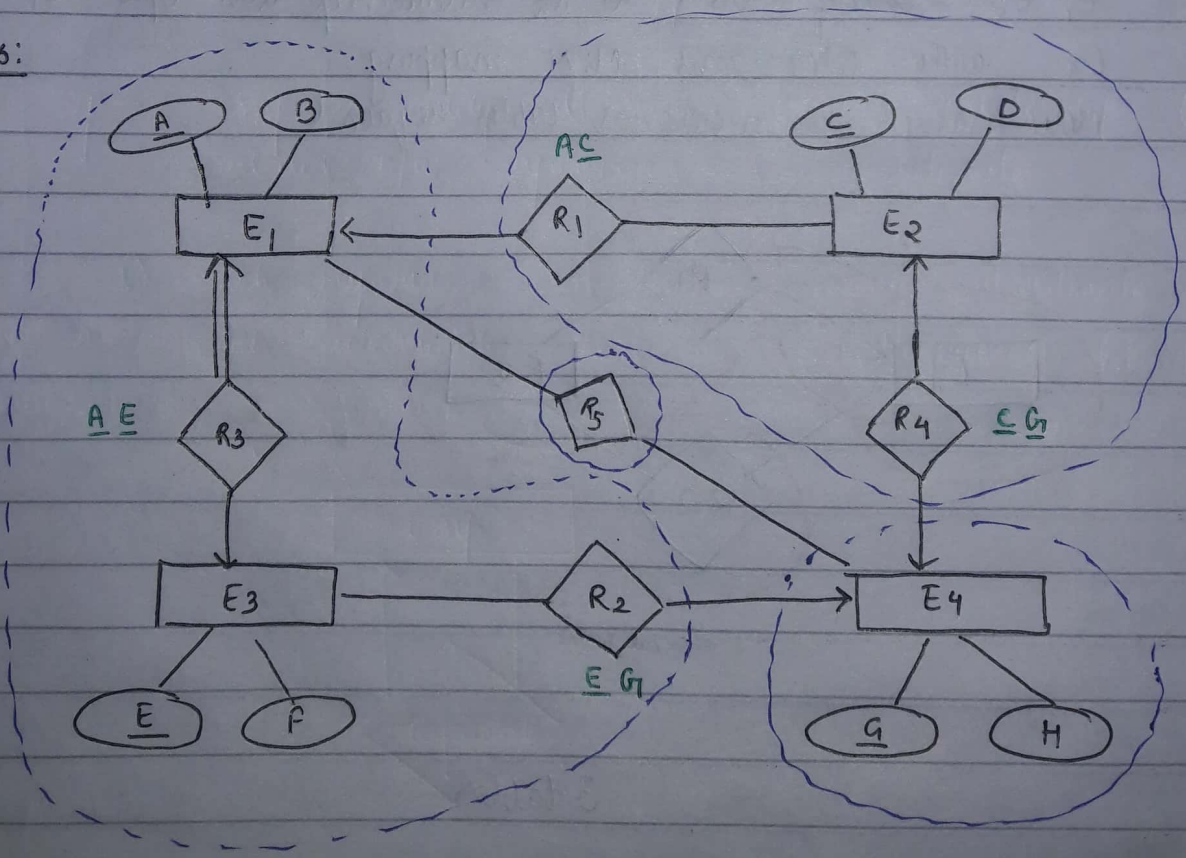
Ques: E_1, E_2, E_3 entity sets
R_1 rel b/w E_1 E_2 with 1:M mapping
R_2 rel b/w E_2 E_3 with M:1 "
R_3 " " E_1 E_3 " 1:1 "
How many min rel table required.





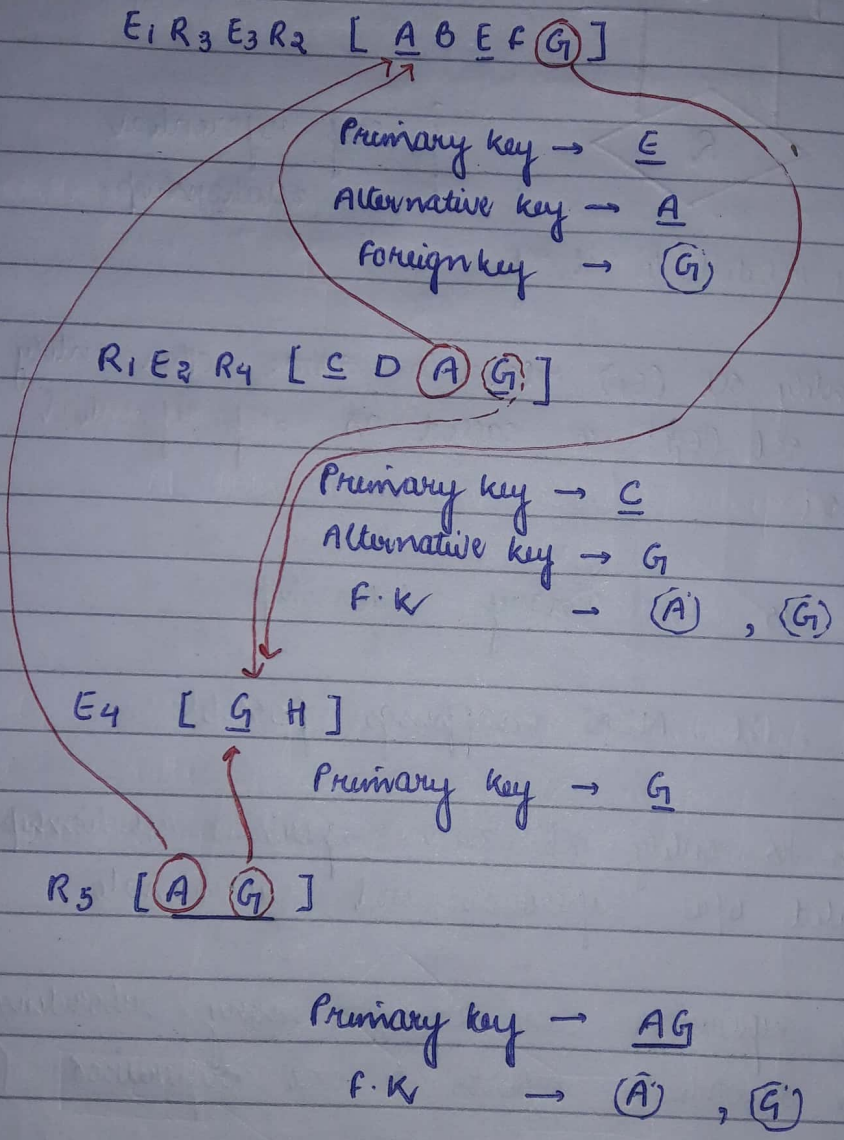
∴ 3 relational tables are required.

Ques:



i) Min rel tables ?
ii) No. of FK in RDBMS design ?
iii) " " attributes in " " ?

Soln:



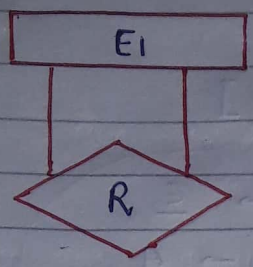
∴

Min rel table req = 4 tables. Ans

No. of F.K = 5 F.K. Ans

Total no. of attribute = 5 + 4 + 2 + 2
= 13 Ans

Self Referential Relationship set —



} recursive entity set

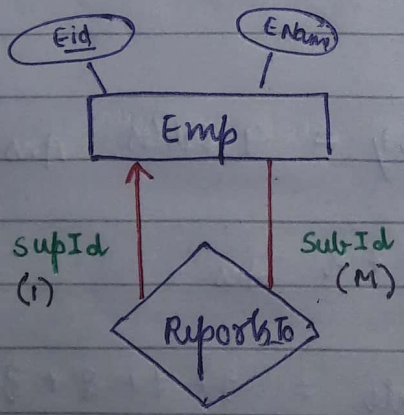
} self referential relationship.

E1 related to itself.

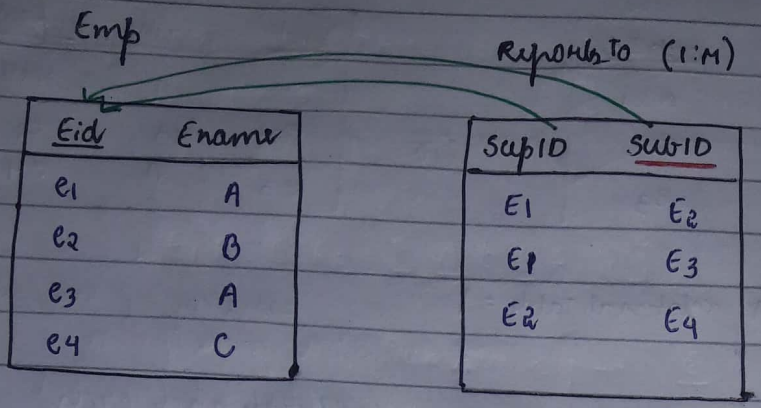
- Entities of entity set (E_1) related to same other entity of same entity set (E_1) is called as self referential relationship set.
- It is same as a binary relationship
- 1:M , M:1 , 1:1 , M:N mappings possible.

Eg – Emp is entity set and reports_To relationship set related b/w supervisor and subordinate.

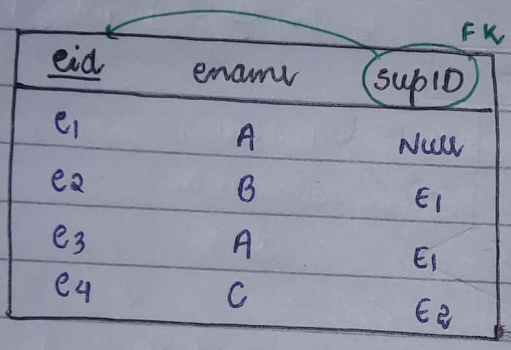
a) Each supervisor can supervise many subordinate and each subordinate reports to one supervisor (1:M)



1:M
binary relation found at many side

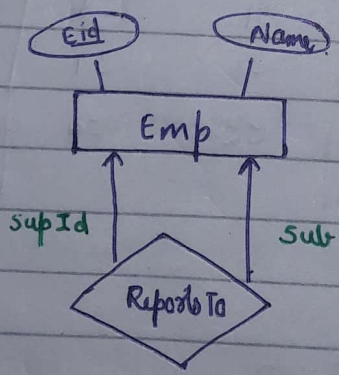


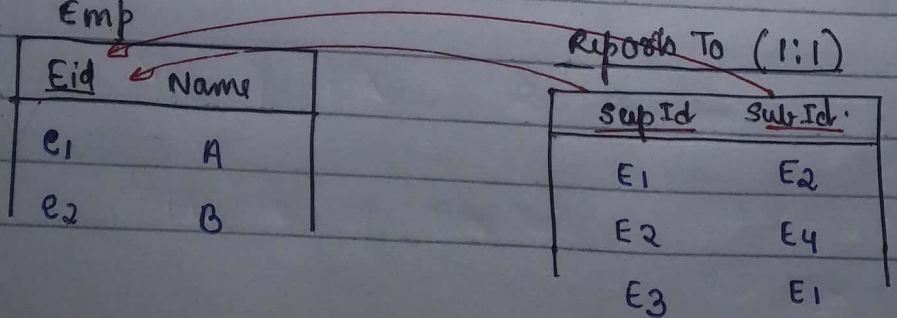
⇓ RDBMS design



∴ Min 1 relational table
and
1 foreign key for ERD

b) each supervisor should supervise one subordinate and each subordinate " report one supervisor (1:1)





Here both the SupId and SubId is unique (as they both can belong only 1 relation

FK

Eid	Ename	SupId
e_1	A	Null
e_2	B	e_1
e_3	A	e_3

cand : {Eid, SupId}

Min relational Table = 1

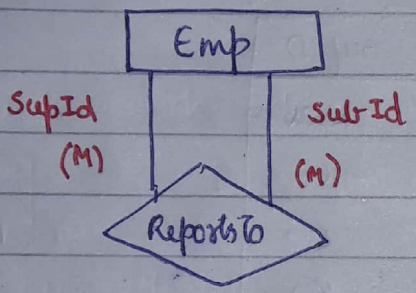
OR

FK

Eid	Ename	subId
e_1	A	Null
e_2	B	e_3
e_3	A	e_4

cand key = {Eid, subId}

c) each supervisor can supervise many subordinate and each subordinate can reporate to many supervisor (M:M)



Emp

Reports To (M:M

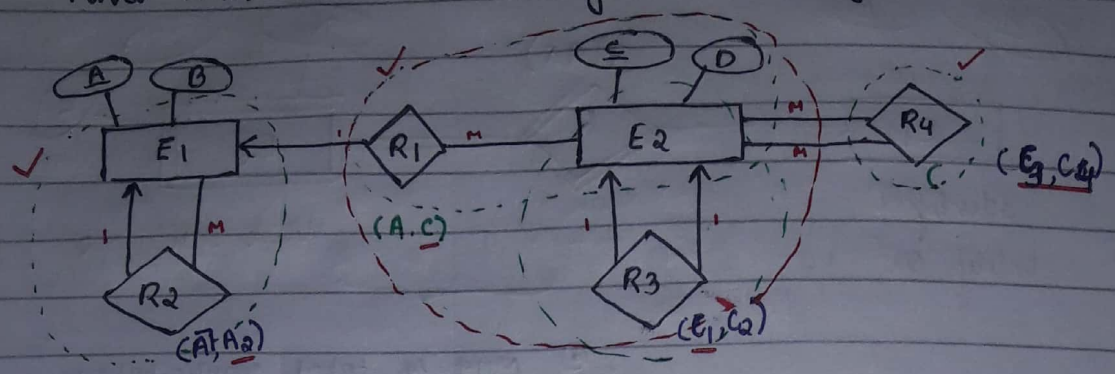
Eid	Ename
e_1	A
e_2	B
e_3	A
e_4	C

SupID	SubID
e_1	e_3
e_1	e_4
e_2	e_4

Min rel table = 2

1:M → combine (key of many side)
1:1 → Both keys
M:1

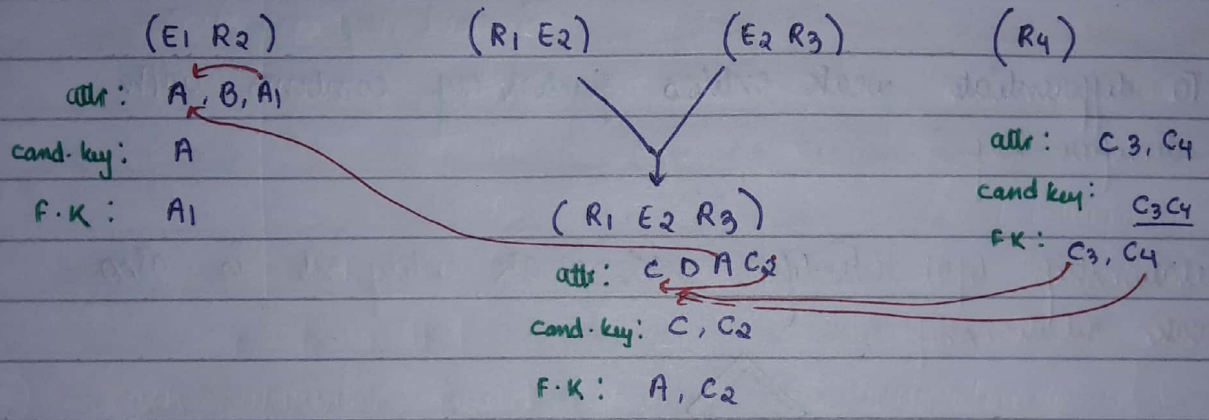
Ques: Find min relational table from ER diagram.



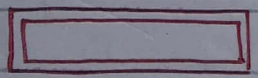
a) Min rel table → 3

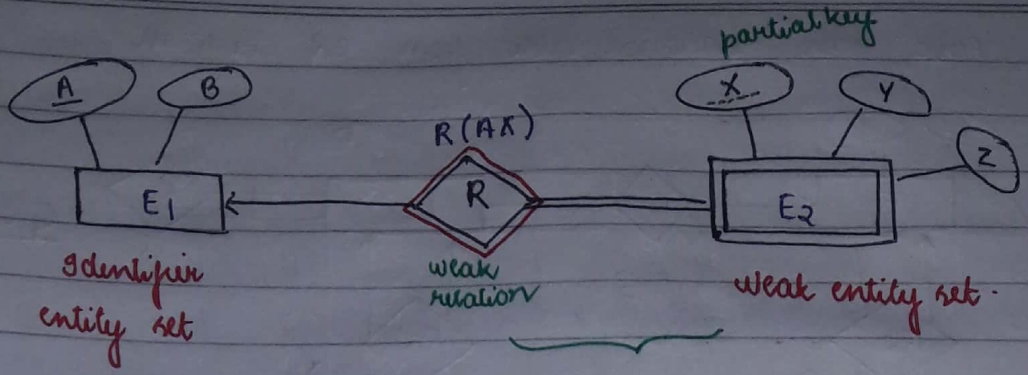
b) No. of FK in RDBMS design (1+2+2) => 5

c) Total no. of attribute in RDBMS design => 9.



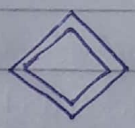
Weak Entity Set —

- Entity set with no key.
- Attributes of entity sets not sufficient to differentiate records.
- Represented by 
- Entities of weak entity set depends on some strong entity set called as identifier (Owner Entity Set)



- Must be total participation at weak entity set end.
- Mapping b/w identifier to weak entity set must be 1 : M

- To differentiate weak entities partial key combines with identifier key
- Relationship b/w identifier and weak entity set is also weak relationship



RDBMS design :

E_1 (A B)

E_2 R (AX Y Z)

Primary key – {AX}
Foreign key (A) ref to E_1

- weak entity set combines with relationship set in RDBMS design which consist of proper candidate key.

For above diagram → 2 rel table.

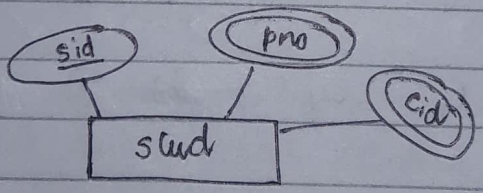
⊙ Multivalued attribute set and weak entity set allowed in ER diagram but, not allowed in relational database (RDBMS)

Multivalued Dependency and 4NF —

→ If relation R is in BCNF
and
redundancy exists in relation R,
then
Atleast two or more multivalued attributes are converted into single valued attribute in R (forms redundancy over MVD's)

→ If relation R in BCNF (i.e no redundancy over FD's)
(and)
Almost one multivalued attribute converted into single valued attribute in R (no redundancy over MVD's)
then, relation R is free from redundancy (i.e No chance of redundancy over FD's and MVD's)

Eg -



Stud (Sid Pno Cid)

S1 P1/P2 C1/C2/C3

S2 P3 C3

⇓

candidate key (Sid Pno Cid)

Stud (Sid Pno Cid)

S_1	P_1	C_1
S_1	P_1	C_2
S_1	P_1	C_3
S_1	P_2	C_1
S_1	P_2	C_2
S_1	P_2	C_3
S_2	P_3	C_3

S_1	P_1
S_1	P_2

same data repeated 3 times

S_1	C_1
S_1	C_2
S_1	C_3

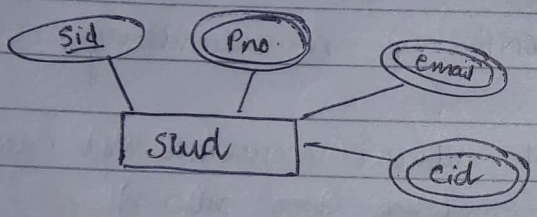
repeated 2 times

⊙ Stud rel is in BCNF.

Redundancy over MVD's can be solved by dividing it in 2 tables

R_1 (Sid Pno) R_2 (Sid Cid)

Eg-



Stud (Sid Pno email Cid)

S_1	P_1/P_2	$e_1/e_2/e_3$	$C_1/C_2/C_3$
S_2	P_3	e_4	C_3

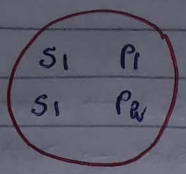
↓

Stud (Sid Pno email Cid)

S_1	P_1	e_1	C_1
S_1			
⋮			
S_1			
S_2	P_3	e_4	C_3

$2 \times 3 \times 3$ => 18 tuples for S_1

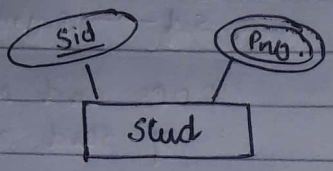
Stud rel in BCNF.



same data repeated 2 times

∴ Redundancy.

Eg-



Stud (Sid Phno)

S1	P1/P2
S2	P2

⇓

Stud (Sid Pno)

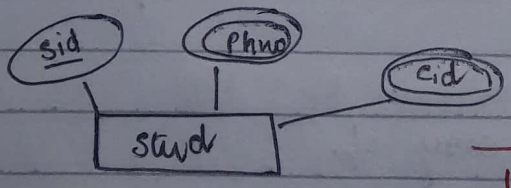
S1	P1
S1	P2
S2	P2

} BCNF relation and. NO redundancy over MVD's

Multivalued dependency — (MVD)

If multivalued attribute (Y), which depends on key attribute (X), if converted into single valued attribute in R, then

X →→ Y MVD exists in R



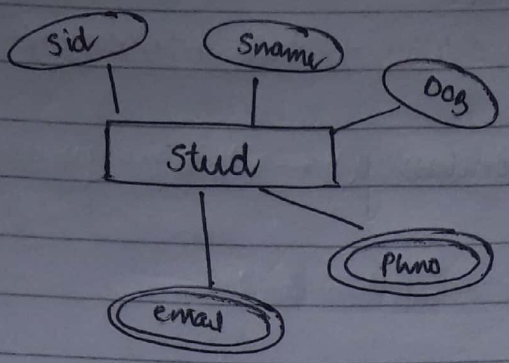
1NF design.

Stud (Sid Phno Cid)

Sid →→ Phno
Sid →→ Cid } MVD

cand key : { Sid Phno Cid }

Eg -



{ Sid → Sname
Sid → DOB
Sid →→ Phno
Sid →→ Email }

FDPs and MVD set of Stud relation.

*** In PSU's

Properties of MVD's —

① Complementation rule :

If $X \rightarrow\rightarrow Y$ is MVD of R, then
$X \rightarrow\rightarrow R-(XUY)$ is also MVD of rel R.

Eg- R(ABCD)
If $A \rightarrow\rightarrow B$ exist
then $A \rightarrow\rightarrow CD$ also exist.

R (A B C D)

4	5	6	7
4	6	7	7
4	5	6	7
4	6	7	7

$A \rightarrow\rightarrow B$ ✓
$A \rightarrow\rightarrow CD$ must also be true.

② Trivial MVD —

$X \rightarrow\rightarrow Y$ is trivial MVD only if
$X \supseteq Y$ (or) $XUY = R$

Eg – stud (Sid Phno)

{ Sid →→ Phno } Trivial MVD → free from redundancy.

Note- No non trivial MVD's possible in relation R(AB) with only 2 attributes

{ Sid →→ Sid, Phno →→ Phno } Trivial MVD

Eg – stud (Sid Pno Cid)

S1	P1	C1	{ Sid →→ Cid }	Non Trivial MVD
S1	P2	C2		
S1	P1	C1	{ Sid →→ Pno }	Non Trivial MVD
S1	P2	C2		

Eg – R(ABCD)

{ AB →→ CD }

Trivial MVD

(3) MVD's not allowed to split over determiner also; also not allowed to merge if determinants are same.

In FD's : If $X \to YZ$ is implied in R, then $X \to Y$, $X \to Z$ also exist in R.

In MVD's : If $X \to\to YZ$ implied in R, then $X \to\to Y$, $X \to\to Z$, may / may not implied in R

Eg – stud (Sid Cno Rno Phno) { Sid →→ CnoRno }

S1 →→ C1R1 | C2R2 | C1R2

{ Sid →→ Cno, Sid →→ Rno } may not in R.

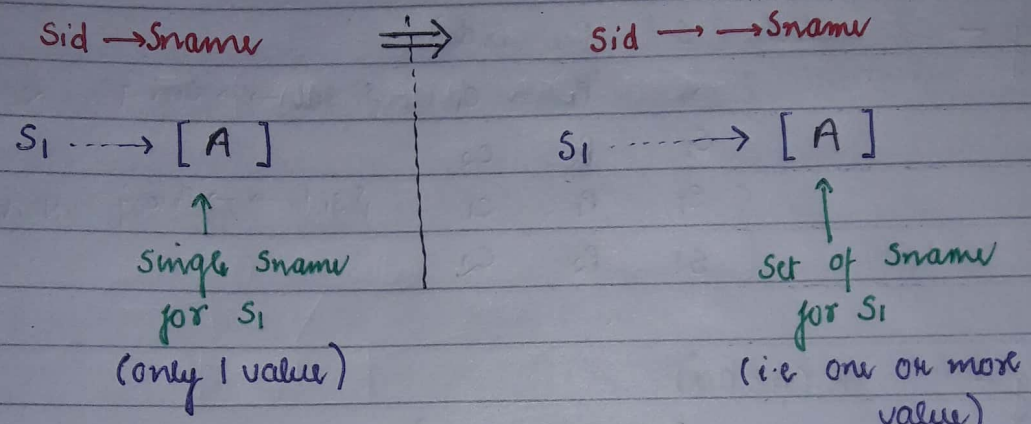
④ Replication Rule –

Every FD ($X \to Y$) can be replaced by MVD ($X \to\to Y$)

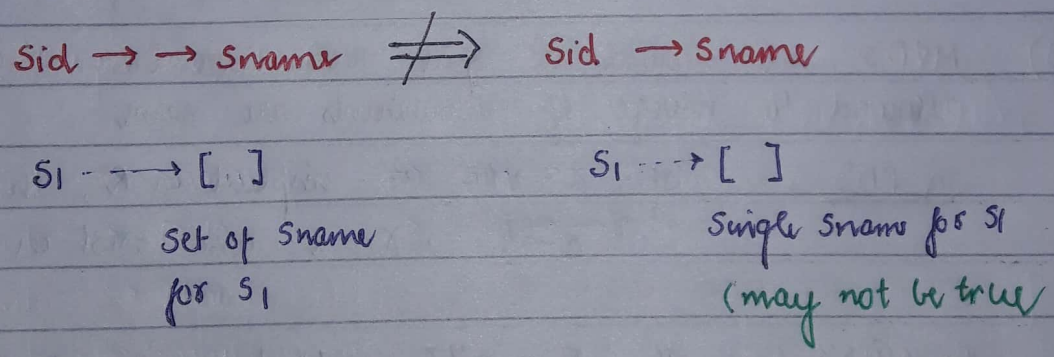
i.e

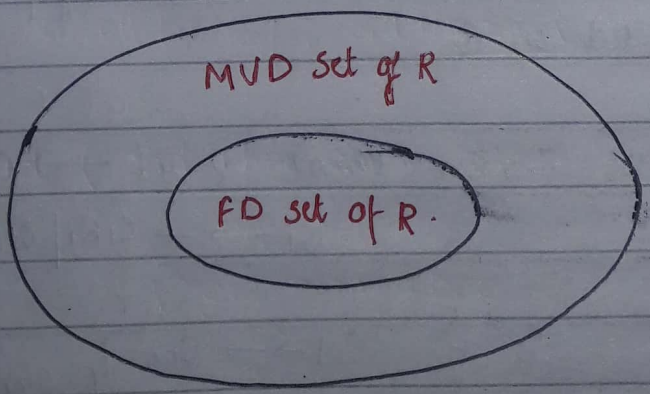
if $X \to Y$, then $X \to\to Y$

Eg –



reverse of it is not allowed.





DRDO

Ques: which is correct ?

1) if $A \to BC$, then $A \to B$, $A \to\to C$
2) if $A \to\to BC$, then $A \to B$, $A \to\to C$
3) if $A \to BC$, then $A \to\to B$, $A \to\to C$
4) if $A \to\to BC$, then $A \to\to B$, $A \to\to C$

(a) 1, 2 (b) 2, 4
(c) ✓ 1, 3 (d) All true

4NF –

definition: Relational schema R is in 4NF iff

(a) every non trivial FD, $X \to Y$ in R with X as superkey

and (b) every non trivial MVD $X \to\to Y$ in R with X as superkey.

sufficient for 4NF as @ is subset of d

$A \to B$, $A \to C$ → convert to MVD

$A \to\to D$ $A \to\to B$
$A \to\to E$ $A \to\to C$

Now check for this

Note –

Attribute closure
Candidate keys identification
Testing of attribute set super key or not

} should use only FD set of relation

Eg- R(ABCD)

$\{A \to B, B \to C, A \to\to D\}$

Not be taken.

$A^+ = \{A, B, C\}$

- candidate key

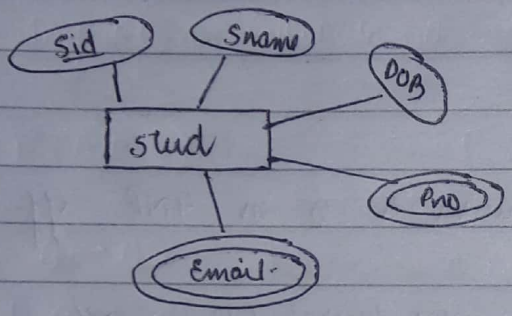
$(AD)^+ = \{ABCD\}$

$\therefore \{AD\}$

- No. of S.K's in R

$\Rightarrow \{AD, ADB, ADC, ABCD\}$

Ques:



Design in INF

Sid → Sname ⇒ Sid →→ Sname
Sid → DOB ⇒ Sid →→ DOB
Sid →→ Phno
Sid →→ Email

candidate key : Sid Phno Email

In INF –

Stud (Sid Phno Email Sname DOB)

$\{$ Sid → Sname DOB , Sid →→ Phno , Sid →→ Email $\}$

Not allowed in 2NF

In 2NF —

We need to decompose it into 2 table

Stud1 (Sid Sname DOB) Stud2 (Sid Phno Email)

Sid → Sname DOB

super key

cand key : Sid

Sid →→ Phno

Sid →→ Email

Non trivial MVD's

Not super key

cand key : Sid Phno Email

∴ Not in 4NF

2NF ✓

3NF ✓

BCNF ✓

4NF ×

Min No. of relational table till BCNF = 2

In 4NF — decomposing it

Stud1 (Sid Sname DOB) Stud (Sid Phno) Stud (Sid email)

Sid → Sname DOB

CK = Sid

Sid →→ Phno

CK = Sid Phno

Sid →→ Email

CK = Sid Email

trivial MVD (∵ No restriction)

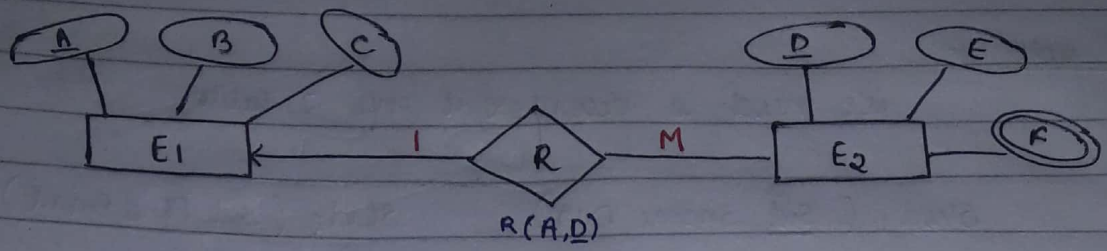
Min No. of relation table for 4NF = 3

Note —

If BCNF rel R (and) atleast 2 MVA stored in R by converting into SVA, then ↓ R not in 4NF

If R is in BCNF (and) Atmost one MVA stored in R by converting into SVA, then ↓ R also in 4NF

Ques:



1) Min relational tables which satisfy 1NF? → 2

2) Min rel table which satisfy 2NF? → 3

Soln:

Rel (R) merges with E_2

and

In RE_2, attribute 'A' must be foreign key ref to E_1

foreign key reference allowed.

E_1 (A B C)

$A \rightarrow BC$

cand key: A

✓ in 4NF

E_2R (DEFA)

$D \rightarrow E$, $D \rightarrow A$ partial depay

$D \rightarrow\!\!\!\!\rightarrow F$

cand key: DF

(Not in 2NF)

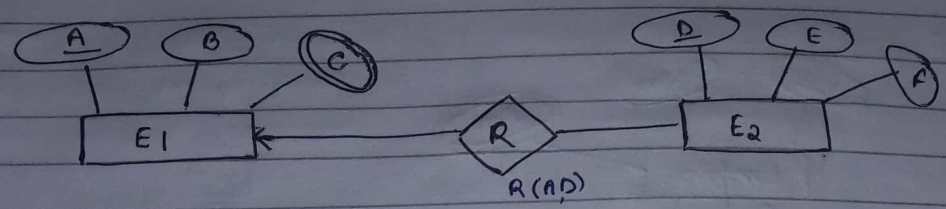
Min 2 Table for 1NF

⇓ decompose

A BC	D AE	DF
$A \rightarrow BC$	$D \rightarrow E$	$D \rightarrow\!\!\!\!\rightarrow f$
	$D \rightarrow A$	

∴ Min rel table for 2NF, 3NF, BCNF, 4NF → 3

Ques:



Min rel req for INF?

Soln:

Rel (R) merge with E_2 and 'A' of rel RE_2 must be foreign key ref to E_1 } design constraint

E_1 [A B C]

$A \to B$

$A \to\to C$

cand. key = AC

RE_2 [D E F A]

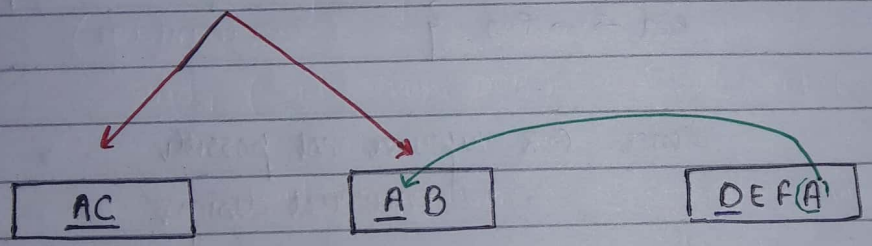
$D \to EFA$

cand. key = D

Foreign key reference Not Possible

as A is not key (constraints are lost)

∴ we cannot do it by 2 table



F.K reference is allowed ✓

$A \to\to C$ $A \to B$ $D \to EFA$

∴ Min 3 tables for INF

H/W → ① in above take 1:1 ?

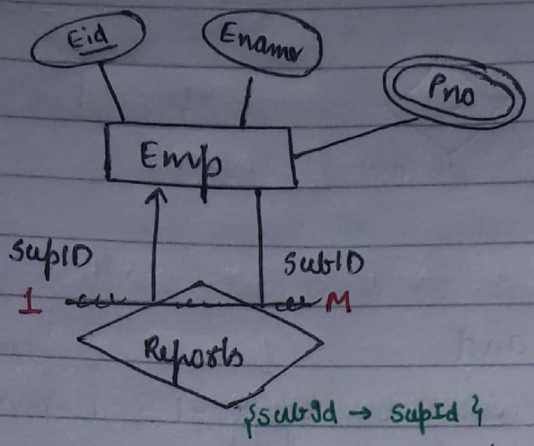
② " " " 1:1 with multivalue both side ?

Note— when converting ER diagram to RDBMS

INF if multivalued attribute exist

4NF if MV attribute does not exist

Ques:



Min rel table required ?

1:M ,

⇒ M side combines

(SubId combine with Eid)

{SubId → SupId}

as SubId is super key it defines all.

Solⁿ:

Emp Reports (Eid , Ename , Pno , supID)

F.K reference not possible (as Eid not a key)

∴

{ eid → Ename
eid → supID
eid →→ Pno }

cand key : {eid Pno}

Since F.K reference not possible

∴ incorrect design

⇓

F.K.

FK allowed

Emp₁ (Eid Pno)

Emp₂ Reports (Eid , Ename SupID)

eid →→ phno

cand key : eid Pno

eid → ename supID

cand key : eid

∴ Min 2 relational table required !

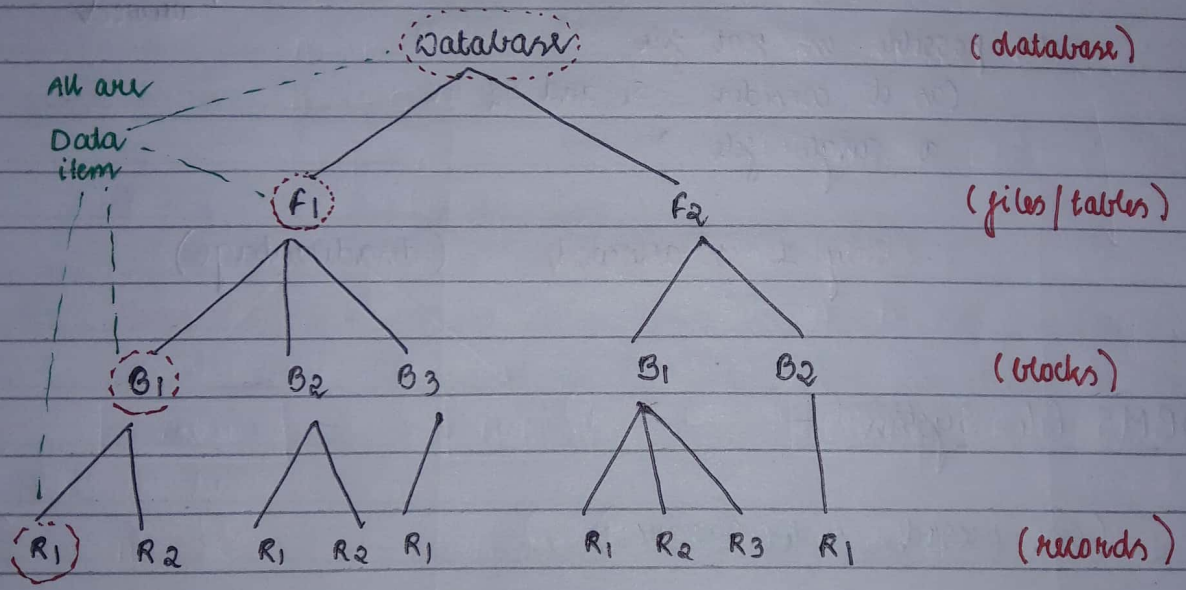
Transactions and Concurrency Control

- **Transaction** is set of logically related operations to perform unit of work.

In OS → Process

In DBMS → Transaction

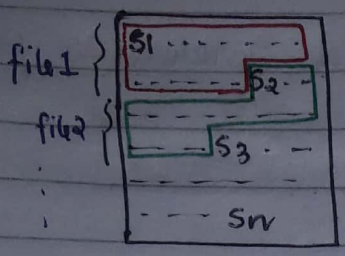
- **Data item** (shared resource) —
  DB element which is required to access by many transaction.



Degree of concurrency — No. of users who can use DB simultaneously (concurrently)

Flat File System (OS file system) ⇓ (1 File is a resource)

DBMS file system



→ but, in flat file it ~~can't~~ see S_1 and S_2 in same file.

In flat file, the data is stored as shown above.

↓

suppose more than 2 update but on different place

↓

U_1: update S_1
U_2: update S_2

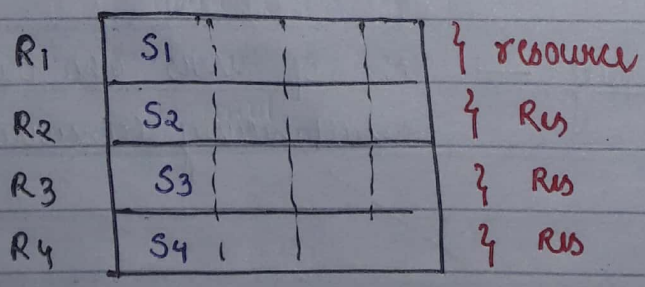
} concurrent execution of U_1, U_2 req not allowed

Not possible in flat file
(as it consider S_1 and S_2 in a single file)

∴ Only 1 is allowed (disadvantage)

DBMS File System —

(one record is a resource)



Since, DBMS point of view, record is a resource

↓

↓

so it allows as many user, as the no. of records

⇓

More degree of concurrency using DBMS file system, because concurrency control over record level.

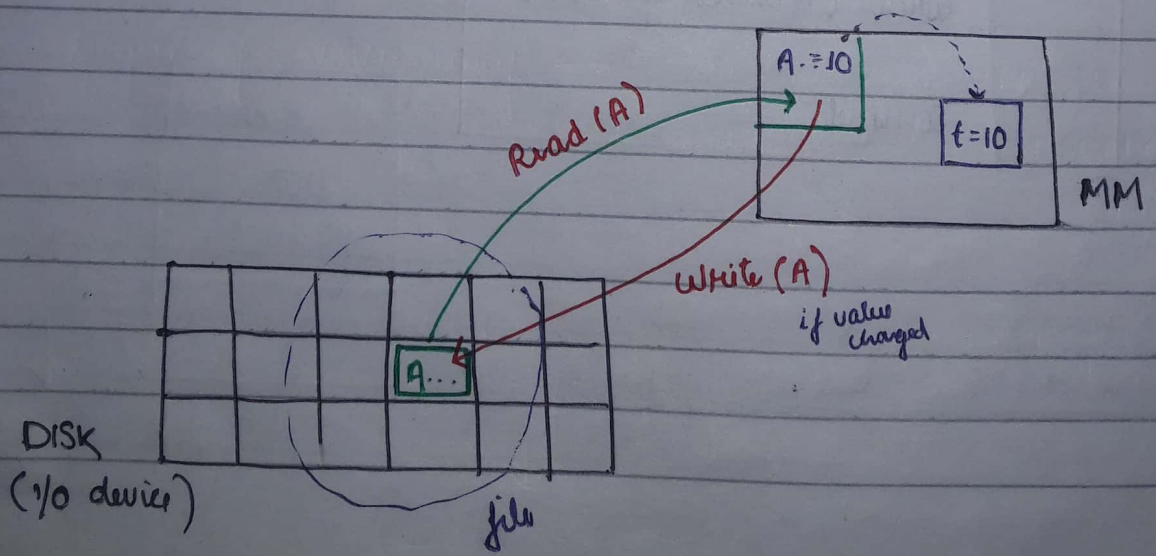
* We cannot do the same thing in OS, as the file structure is not under user. (fixed)

Main Operations in Transaction —

A : data item

① Read (A) — Access data item 'A' from DB file (DISK) to programmed variable (MM) in order to use current value of 'A' in transaction logic.

② Write (A) — Modification of data item 'A' in DB file.



→ Account (Cid bal)

Trans (T_i) : Transfer 500 from customer C_1 to C_2

SQL queries →

```
begin trans Ti
    update account set bal = bal - 500
        where Cid = C1;
    update account set bal = bal + 500
        where Cid = C2;
end trans
```

⇓ SQL parser executes

Trans (T_i)

Read (Balance of C_1) — A

A = A - 500

Write (Balance of C_1) — A

Read (Balance of C_2) — B

B = B + 500

Write (Balance of C_2) — B ;

Commit ;

← Transaction (various read and write together)

Trans (T_2): set bal of C_1 C_2 as 10,000

```
Update account set bal = 10000 where Cid = C1 ;
update account set bal = 10.000 where Cid = C2 ;
```

⇓

Trans (T)

write (bal = 10000 of C_1) ← A

write (bal = 10000 of C_2) ← B

commit

Blind write (writing back in DB without any knowledge of prev value)

Trans (T_3):

```
R (A)
R (B)
W (B)
W (C)
W (D)
commit
```

Blind write

ACID Properties —

To preserve integrity (correctness) each transaction must satisfy ACID property.

A : Atomicity	}	Recovery Management component of DBMS s/w takes care of A and D
D : Durability	}	
I : Isolation	}	
C : consistency	}	

DBMS s/w

- A : Atomicity
- D : Durability

} Recovery management component of DBMS s/w takes care of atomicity and durability

- I : Isolation

Concurrency control component of DBMS s/w is responsible for isolation

DBA user

- C : Consistency

USER [DBA or DB developer] is responsible for consistency

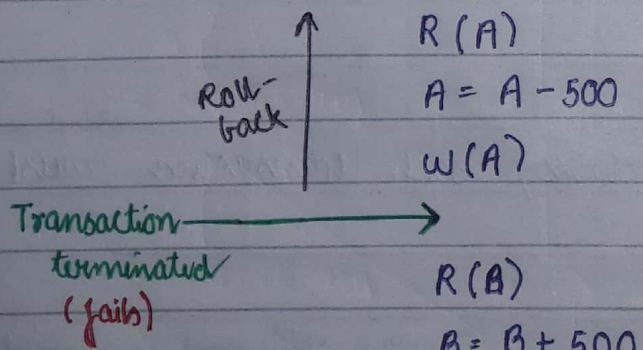
Atomicity –

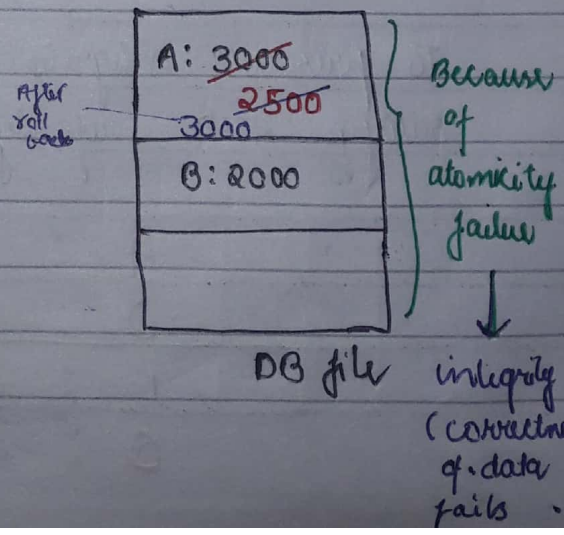
(Execute all operations of transaction including commit)

(OR)

(Execute none of the operation of transaction) by the time of transaction termination.

Eg- Trans (T_i) : transfer 500 from A to B





↓ Recovery manager

Roll backs all the transaction done by T_i

⇓

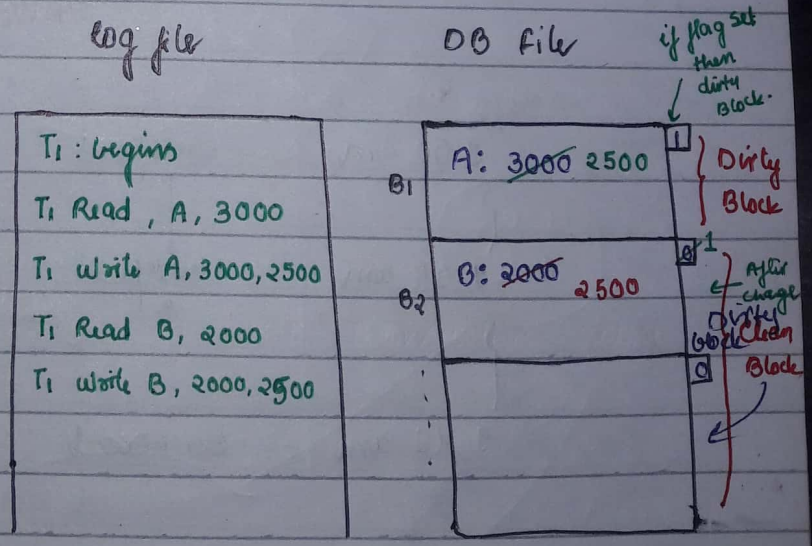
If Rollback (Undo oper^n) is successful

⇓

'Atomicity Preserved'

- Recovery Management component should roll back, if transaction fails anywhere before commit.
- Roll back (Abort) — Undo modification of database file which are done by failure transaction.
- Transaction log — File maintained by recovery management component (RMC) in DISK (secondary memory) to record all the activities of transaction.

Transaction (T_i)	log file	DB File
R(A)	T_i : begins	
A = A - 500	T_i Read, A, 3000	
W(A)	T_i Write A, 3000, 2500	
R(B)	T_i Read B, 2000	
B = B + 500	T_i Write B, 2000, 2500	
W(B)		
Commit		



Dirty Block — updated by uncommitted transaction.

- **Redo operations** —

→ Re-performs all the modification of database file because of transaction commit (i.e after commit)

→ Clean all log entries of commited transaction.

→ Redo is not done after every commit, but after checkpoint

commit 1 → redo (T_1)

commit 2 → redo (T_2)

commit 3 → redo (T_3)

commit 4 → redo (T_4)

because of many redo operation, "Throughput of system decreases"

⇓ To avoid this problem 'Consistency check point' is introduced

9:00 am — checkpoint

9:05 am — checkpoint

9:10 am — checkpoint

"All commit transaction until previous checkpoint performs redo"

FAILURE → 9:14 am

(Need to Recover)

→ Roll back till last checkpoint

- If checkpoint issued, all commited transactions until previous checkpoint should perform redo and clean log entries of commited transaction.
- If system failure happen, required oper^n to recover are

  ① All commited transaction until prev checkpoint will perform Redo and,

  ② All uncommited transaction in entire system will perform Undo.

  ③ Clean log entries.

Ques: Consider the log file entries

	1.	T_1 begin
	2.	T_1 W, A, 10, 20
	3.	T_2 begin
	4.	T_2 W, B, 0, 5
	5.	T_2 commit
	6.	T_3 begin
In entire sys whoever no commits go for Undo T_1, T_3, T_5	7.	T_3 W, B, 5, 10
	8.	checkpoint
until prev check point who ever commited will redo (T_4)	9.	T_4 begin
	10.	T_5 begin
	11.	T_4 W, C, 0, 100
	12.	T_4 Commit
	13.	failure

T_1 T_2 ✓ T_3 T_4 (commit) T_5

Commited → redo of T_2 done

What are required oper^n performed to recover from failure

a) T_2 T_4 redo and T_1 T_3 T_5 Undo

b) T_4 redo, T_5 T_1 T_3 undo

c) T_1 T_2 T_3 T_4 T_5 Undo

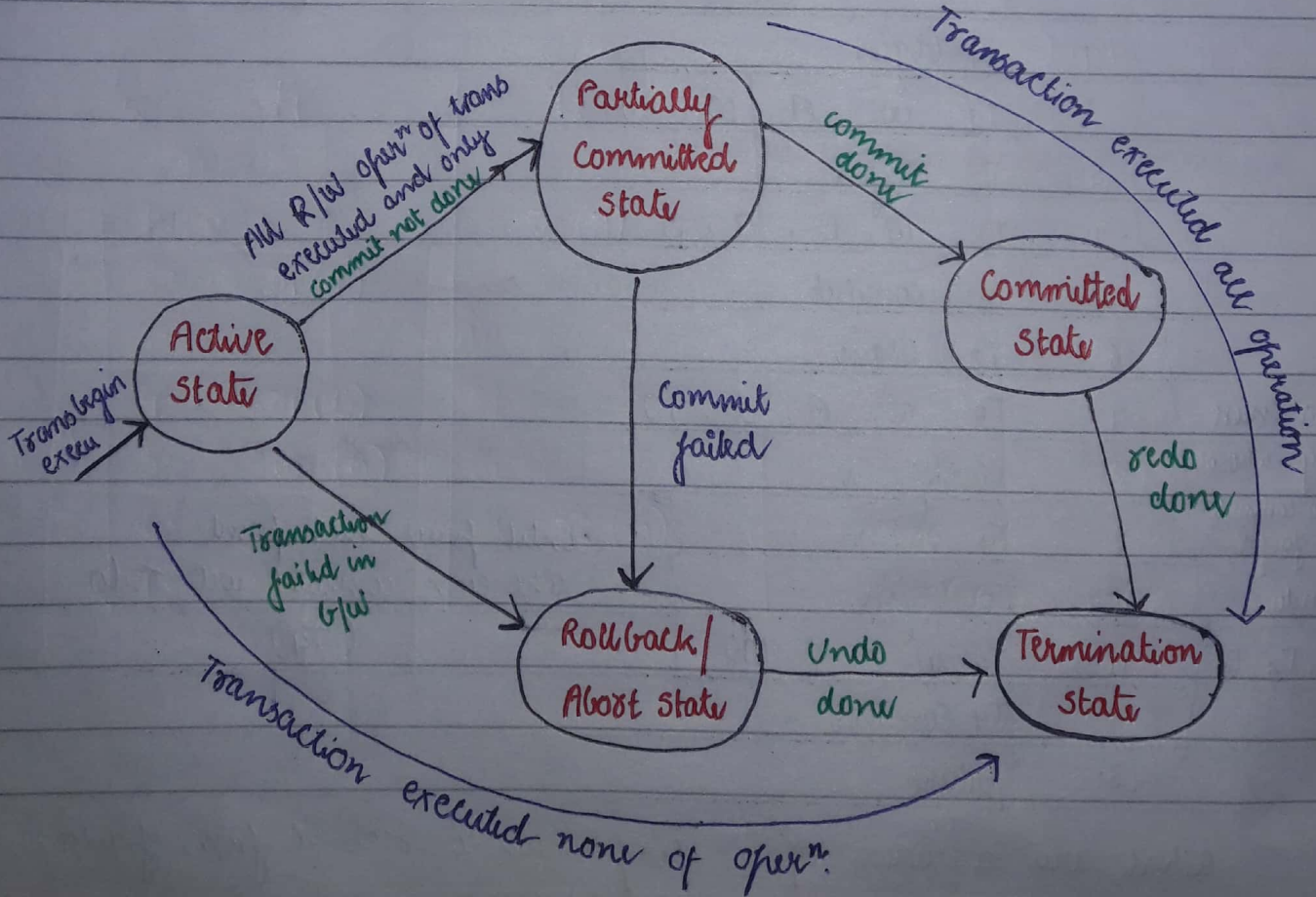
d) T_2 redo and T_1, T_3 T_4 T_5 Undo

redo : T_4

Undo : T_1, T_3, T_5

Note - Before checkpoint T_2 completed Redo i.e commit.

Transaction States –



Durability —

- Transaction should able to recover under any case of failure.

- Transaction fails because of :

  ① Power failure

  ② S/W Crash
  - OS restarted
  - DBMS restarted

  ③ OS / DBMS concurrency control may kill transaction.

  ④ H/W Crash (DISK Crash)

should be able to roll back 'durable'

if log file and state of each trans maintained in DISK

RAID architecture of disk design is used

→

RAID : 0 — data in main disk — Not durable

RAID : 1 — 2 disk

Mirror Image of Disk

.
.
.
.

Isolation —

(Maintained by Concurrency Control Component)

Isolation means concurrent execution of 2 or more transactions result must be equal to result of some serial schedule.

Schedule : Time order execution sequence of 2 or more transaction.

Eg —

T_1 : $r_1(A)$ $r_1(B)$ $w_1(B)$

T_2 : $r_2(A)$ $r_2(B)$

T_3 : $w_3(A)$ $w_3(B)$

⇓ Concurrent Representation (Schedule)

S	T_1	T_2	T_3
1.		$r_2(A)$	
2.	$r_1(A)$		
3.			$w_3(A)$
4.	$r_1(B)$		
5.	$w_1(B)$		
6.		$r_2(B)$	
7.			$w_3(B)$

Time order diagram ↓

Types of schedule —

① Serial schedule

②

→ **Serial Schedule :**

- Transaction should execute one after other.
- After the commit of current executing transaction, allow the beginning (start) of other transaction.

Eg — T_1 : Transfer 500 from A to B

T_2 : display total bal of A and B

S_1 :

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	
	← only after commit, it can start.
	$r_2(A)$
	$r_2(B)$
	disp (A+B)
	commit

OR

S_2 :

T_1	T_2
	$r_2(A)$
	$r_2(B)$
	dis (A+B)
	commit
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	

Advantage —

- Serial schedules gaurenter produces correct result (integrity gaurenteed) as no resource sharing.

Disadvantage —

- It allows transaction to execute serially only.
- less degree of concurrency.
- Throughput of system is less.

→ Concurrent Schedule

Transactions can execute concurrently / simultaneously / interleaved order.

Eg -

S_3:

T_1	T_2
3000 $r_1(A)$	
2500 $w_1(A)$	
	$r_2(A)$ 2500
	$r_2(B)$ 2000
	disp(A+B) 4500
	commit
$r_1(B)$	
$w_1(B)$	
commit	

Incorrect

S_4:

T_1	T_2
3000 $r_1(A)$	
2500 $w_1(A)$	
	$r_2(A)$ 2500
2000 $r_1(B)$	
2500 $w_1(B)$	
commit	
	$r_2(B)$ 2500
	disp(A+B) 5000
	commit

Correct

- Concurrent schedule are both serial (non serial schedule)

On executing S_3 —

$A = 3000$
$B = 2000$

→ Schedule not equal to $T_1 \rightarrow T_2$ serial and $\neq$ $T_2 \rightarrow T_1$ serial
(Isolation Fails)

disp (A+B) = 4500
(incorrect result)

On executing S_4 —

$A = 3000$
$B = 2000$

→ Schedule equal to $T_1 \rightarrow T_2$ serial. (S_1)
(Isolation satisfied)

disp (A+B) = 5000
(correct result)

∴ Result of concurrent schedule may or correct or incorrect.

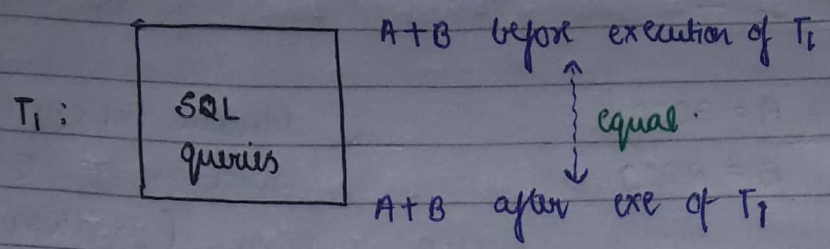
14/11/17

Consistency —

(User responsible for consistency)

- DB operations requested by user [SQL queries / transaction oper^n) must be logically correct
- Consistency depends on transaction to transaction.

T_1 : Transfer 500 from A to B



⑥

A
D
I
C

} Recovery Manager
} Concurrency controller
} USER

} responsible for integrity (correctness of trans result)

Schedule Classification

- Serializability (Isolation must satisfy)
  - ⓐ Conflict serializable schedule
  - ⓑ View serializable schedule

  → Serializability Testing (Isolation testing)
- Recoverable Schedules (atomicity and durability must satisfy)

Other Problems of Concurrent Execution —

Even if isolation is preserved, these problems can occur:

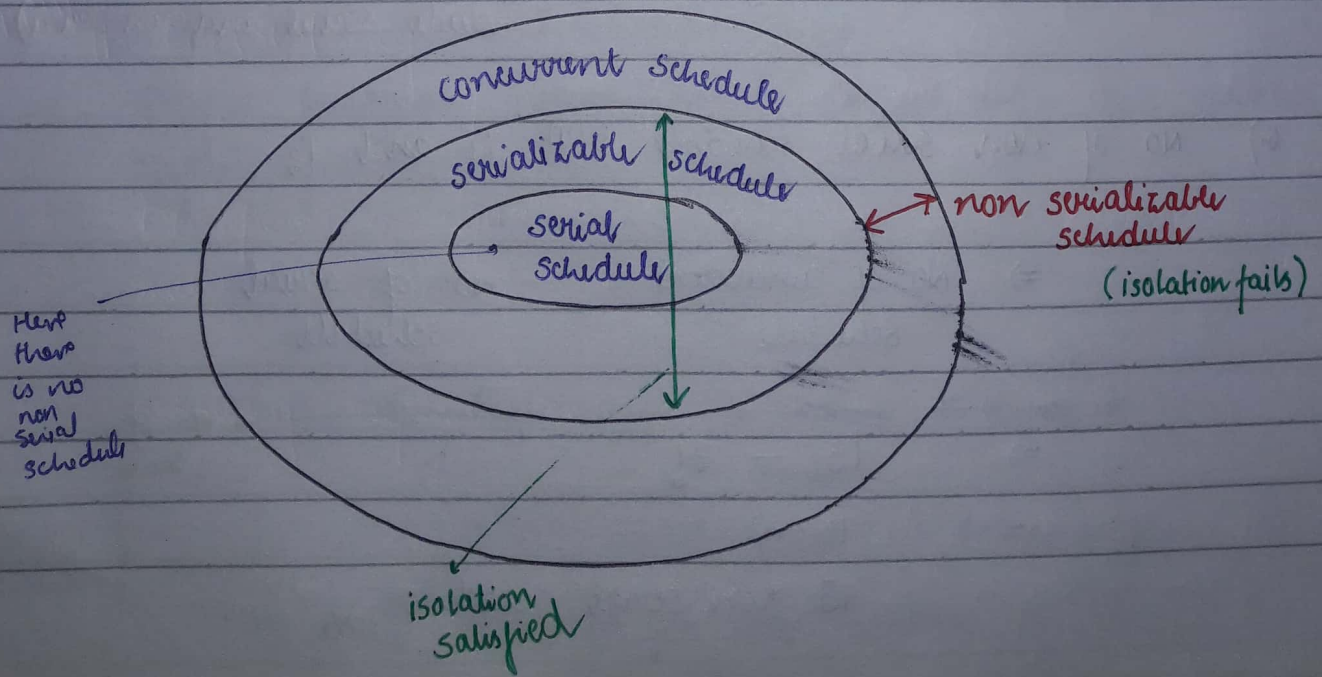
1. Irrecoverable schedule Problem
2. Cascading Rollback problem
3. Lost Update problem.

⇓ To avoid above problem solⁿ are —

1. Recoverable schedule (for irrecoverable sche prob)
2. Cascadeless Rollback Schedule (for cascading roll back)
3. Strict Recoverable Schedule (for lost update prob)

Serializable Schedule —

Schedule (s) are serializable iff some serial schedules(s) equal to the schedule (s) i.e preserve isolation.



Ques: T_1 T_2 T_3 ---- T_n transactions.

No. of serial schedule possible = $n!$

Ques: T_1 : $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$ → All should execute in same order

T_2 : $r_2(A)$ $r_2(B)$ → "

$^5C_2$

a) No. of concurrent schedule b/w T_1 and T_2

$^6C_4 \times {}^2C_2$

$\Rightarrow \frac{6!}{2!4!}$

$\Rightarrow 15$

Soln:

	T_1	T_2
1.		
2.		
3.		
4.		
5.		
6.		

T_1 → 6 places, 4 oper arranged in $^6C_4$ way

$= {}^6C_4 \cdot {}^2C_2$

$= 15 \times 1$

$= 15$ concurrent schedule Ans

(include serial and non serial)

b) No. of non serial schedule b/w T_1 and T_2

$\Rightarrow$ No. of concurrent schedule − No. of serial schedule

$\Rightarrow 15 - 2!$ → No. of trans.

$= 15 - 2$

$= 13$ non serial schedule Ans

Ques: T_1, T_2 transaction with n, m operation each.
No. of concurrent schedule.

For T_1, T_2 $\Rightarrow {}^{n+m}C_n$

$\Rightarrow \frac{(n+m)!}{n! \cdot m!}$ Ans

Ques: T_1, T_2, T_3 transaction with n, m, p operation each.

No. of concurrent schedule ?

$= {}^{(n+m+p)}C_n \cdot {}^{(m+p)}C_m \cdot {}^{p}C_p$

$= {}^{(n+m+p)}C_n \cdot {}^{(m+p)}C_m$

$= \frac{(n+m+p)!}{n! \cdot m! \cdot p!}$ Ans

1) Conflict seriatizable Schedule —

→ **Topological Order:** (Greedy Algo)

Graph traversal algorithm applicable only for directed acyclic graph (DAG).

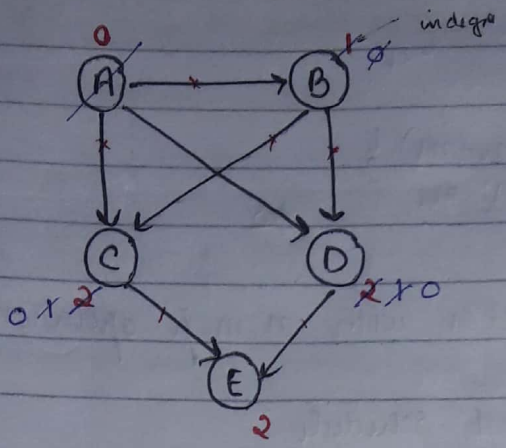
Procedure —

① Visit vertex (v) whose in-degree is "0"
and
delete visited vertex (v) from graph

② Repeat ① for all vertices of graph

Topological order : visited order of vertices.

Eg -

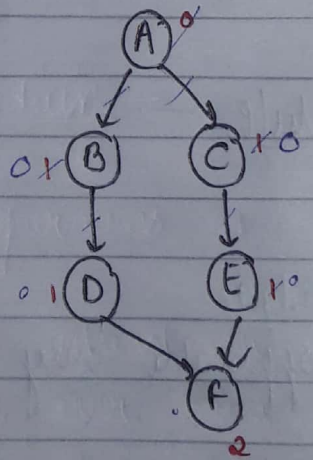


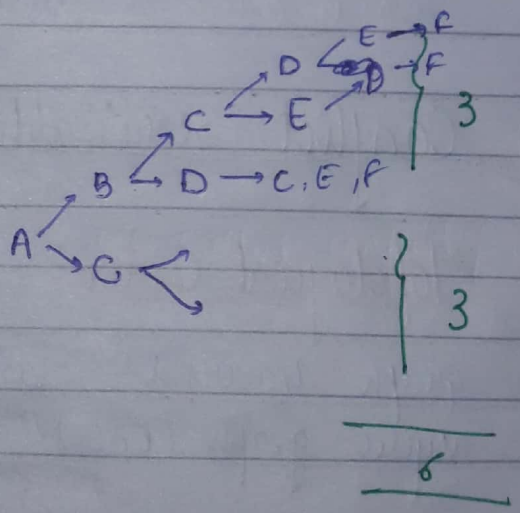
visit indegree 0 and delete its outgoing edges; repeat

A, B → C, D, E
A, B → D, C, E

(2 topological order)

Ques: How many no. of topological order possible





A B D F

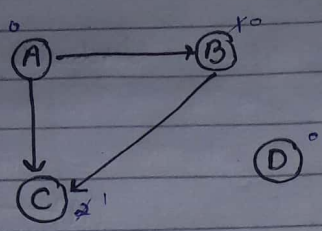
B → before D
C → before E
$^4C_2$ ways

$^2C_2$

$\Rightarrow {}^4C_2 \cdot {}^2C_2$

$\Rightarrow 6$ Ans

Ques:

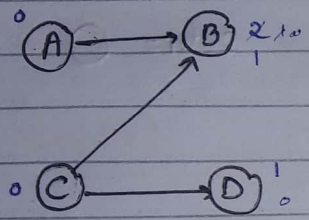


D → B → C
A → B → C → D
A → B → D → C
D A B C

Find no. of topological order.

4

Ques:



No. of topological order.

A C → B → D
A C → D → B
C → A → B → D
C → A → D → B
C → D → A → B

5

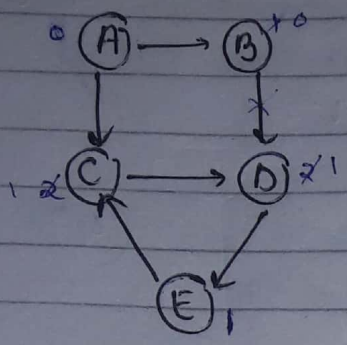
Ques: No. of Topological order of null graph with n vertices. (0 edges)

$n!$

0

6

Ques: Cyclic graph

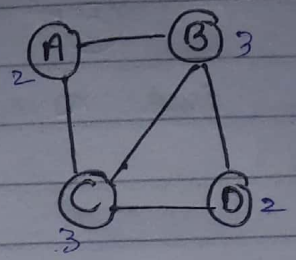


No. of Topolo order ?

order = 0

A B

Undirected graph.



X
Not Possible for undirected

order = 0

Not Possible.
for cyclic graph

Conflict Pair —

Pair of operations in schedule (S) are called conflict pair iff.

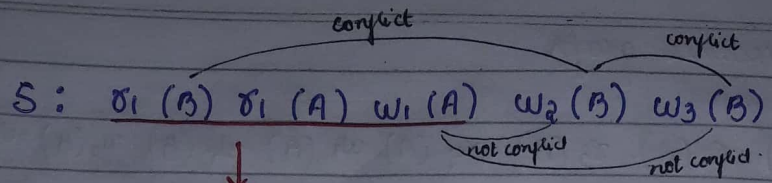
1. Atleast one write operation and
2. over same data item and
3. from different transaction

S: $r_i(A)$ ---- $w_j(A)$
S: $w_i(A)$ ---- $r_j(A)$
S: $w_i(A)$ ---- $w_j(A)$

} conflict Pair

S: $r_i(A)$ ---- $r_j(A)$
S: $r_i(A)/w_i(A)$ ---- $r_j(B)/w_j(B)$

} Not conflict Pair



↓

oprn within transaction should execute in same order

∴ r1(B) r1(A) w1(A)

↓

"Neither conflict nor not conflict Pairs"

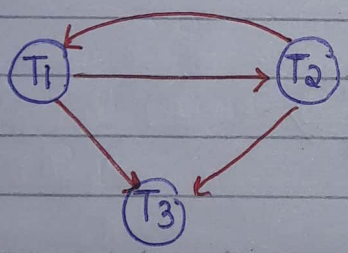
Precedence Graph —

Vertices (V) : Transaction of schedule.

Edges (E) : conflict Pair Precedences

Eg -

S: r1(A) r2(B) w2(A) w1(A) w2(B) w3(B) w3(A)



1) Highest No. is the no. of transact.

2) Take one and see the conflict

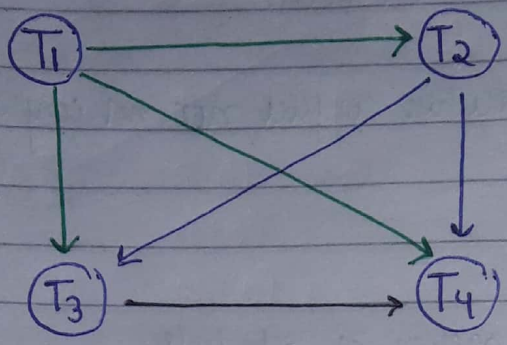
r1(A)

saw A write will be seen if no. is change conflict

Ques: Draw precedence graph

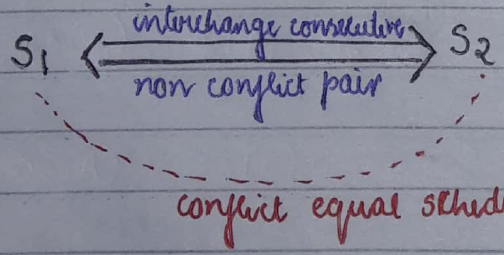
S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

No conflict for read-read.

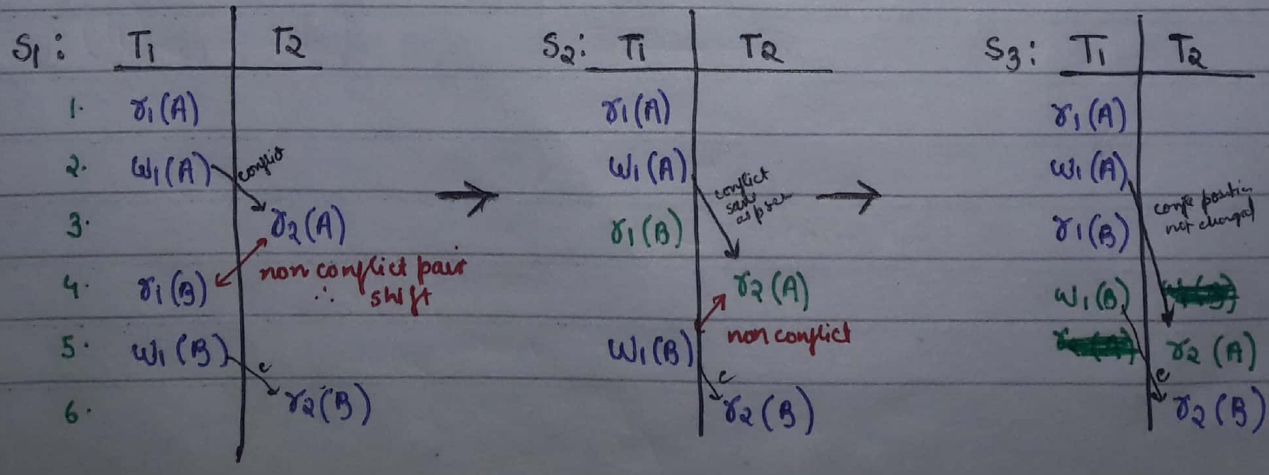


Conflict Equal Schedules —

S_1, S_2 are conflict equal schedules iff S_2 is derived by interchange of consecutive non conflict of schedule S_1



Eg –



Eg -

T_1	T_2
$r_1(A)$	
$r_1(B)$	
	$w_2(B)$

$T_1 \rightarrow T_2$

T_1	T_2
$r_1(B)$	
$r_1(A)$	
	$w_2(B)$

$T_1 \rightarrow T_2$

Not conflict equal

Note:-

→ The precedence graph of all the conflict equal schedules are same. but not vice versa.

S_1: $T_1 \rightarrow T_2$ S_2: $T_1 \rightarrow T_2$ S_3: $T_1 \rightarrow T_2$

Ques: S_1 : $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

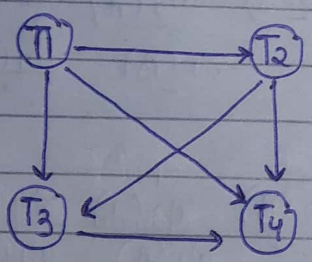
S_2 : $r_2(A)$ $r_1(A)$ $w_1(B)$ $w_2(B)$ $r_3(A)$ $w_3(B)$ $r_4(A)$ $w_4(B)$

conflict pair

S_3 : $r_3(A)$ $r_4(A)$ $r_1(A)$ $w_1(B)$ $r_2(A)$ $w_2(B)$ $r_4(A)$ $w_4(B)$ $w_3(B)$

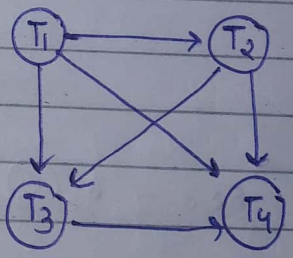
what are conflict equal schedule.

S_1 :

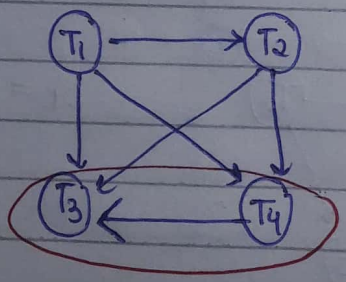


=

S_2 :



S_3 :



S_1 and S_2 are conflict equal schedule

$S_1 = S_2 \neq S_3$

For same precedence graph, does not mean conflict equal

S_1 S_2 are conflict equal iff

① Each transaction of S_1 must be exactly same in S_2
(i.e no. of operations same and sequence same)

S_1 ... T_i ...	S_2 ... T_i ...
$r_i(A)$	$r_i(A)$
$r_i(B)$	$r_i(B)$
$w_i(B)$	$w_i(B)$

and ② Every conflict pair precedence of S_1 S_2 must be same.

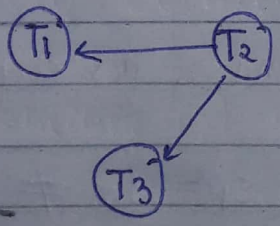
Page ⑦

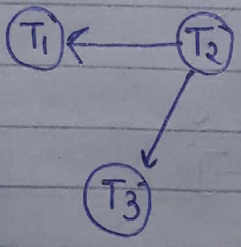
S_1 : $r_2(A)$ $w_2(A)$ $r_3(C)$ $w_2(B)$ $w_3(A)$ $w_3(C)$ $r_1(A)$
$r_1(B)$ $w_1(A)$ $w_1(B)$

S_2 : $r_3(C)$ $r_2(A)$ $w_2(A)$ $w_2(B)$ $w_3(A)$ $r_1(A)$ $r_1(B)$
$w_1(A)$ $w_1(B)$ $w_3(C)$

S_3 : $r_2(A)$ $r_3(C)$ $w_3(A)$ $w_2(A)$ $w_2(B)$ $w_3(C)$ $r_1(A)$
$r_1(B)$ $w_1(A)$ $w_1(B)$

Soln:





$S_1 = S_2 \neq S_3$

(d)

S

If precedence same then S_1, S_2 may or may not conflict equal (as it does not show all conflict pair)

Conflict Serializable Schedule —

Schedule (S) is conflict serializable schedule iff some serial schedule (s) conflict equal to schedule S

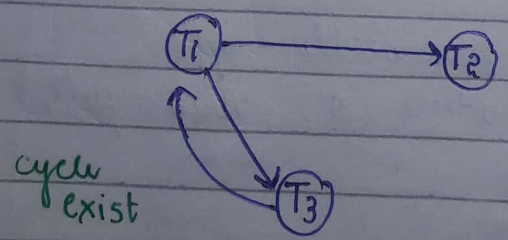
[S : schedule] ⟺ (conflict equal) [S' : serial schedule]

(Precedence graph of schedule (S) must be acyclic) ⟺ (FOR conflict equal) (Precedence graph of serial schedule (S') is always acyclic.)

Testing Condition of Conflict Serializable Schedule —

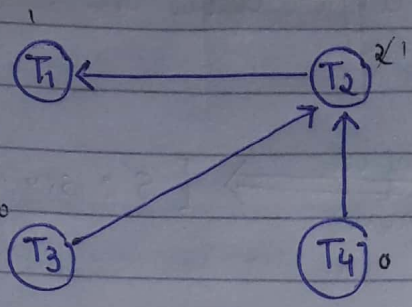
Schedule (S) is conflict serializable schedule iff precedence graph of schedule (S) is acyclic
and
conflict equal serial schedule are topological order of precedence graph.

Eg — S : $r_1(A)\ w_1(B)\ r_2(B)\ r_3(C)\ w_3(C)\ w_3(A)\ w_1(C)$



cycle exist

∴ S is not conflict serializable schedule
(No serial schedule conflict equal to S



acyclic
(i.e conflict serializable schedule)

Now, To compute conflict equal serial

Finding Precedence.

2 serial schedule exist { $T_4\ T_3\ T_2\ T_1$; $T_3\ T_4\ T_2\ T_1$ } which is conflict equal to given S

order = 2

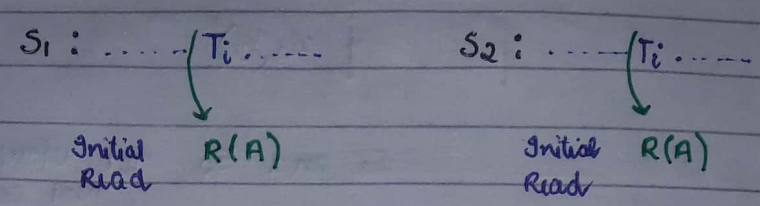
No. of serial schedule conflict equal to sch (S)	=	No. of topological order of schedule (S) precedence graph

② View Serializable Schedule –

Schedule (S) is view serializable iff some serial schedule (S') view equal to schedule (S)

S_1 S_2 are View Equal iff —

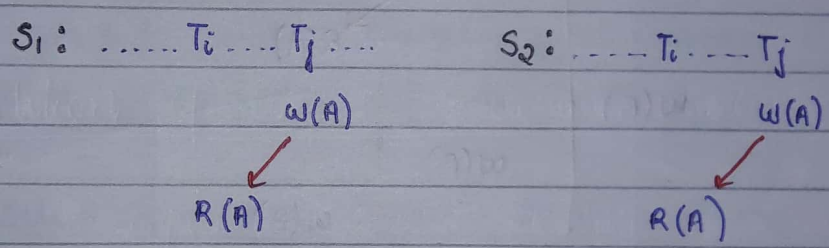
① Initial Read:



Every initial read of S_1 must also be initial read in S_2

Initial read means before read of A, R(A) there is no W(A).

and ② Updated Read:



Every updated read of S_1 must be same updated read in S_2.

and ③ Final Write :

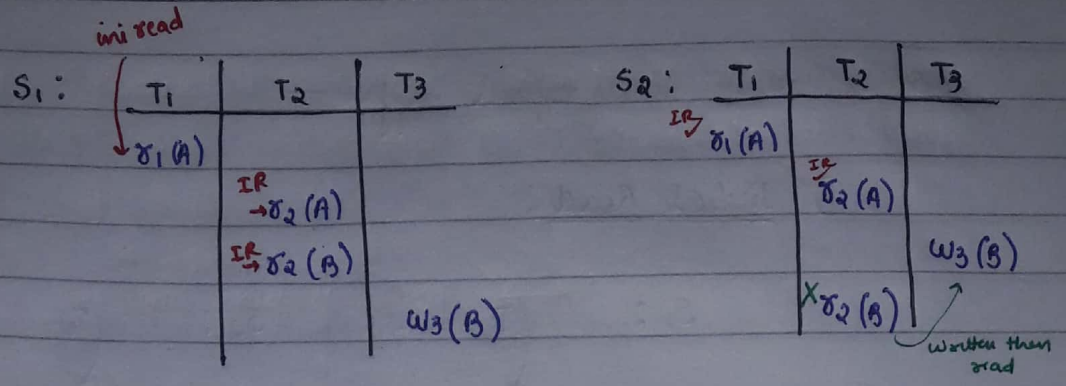
Final write of A (last write) → W(A)

Final write of A → W(A)

Every final writes of S_1 must be same in S_2.

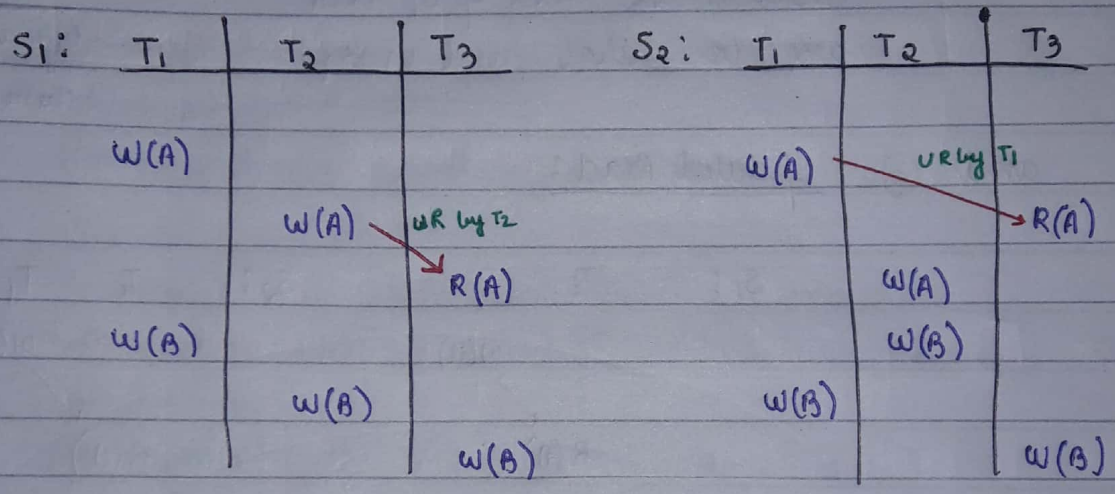
IR - initial read UR - update read

Eg -



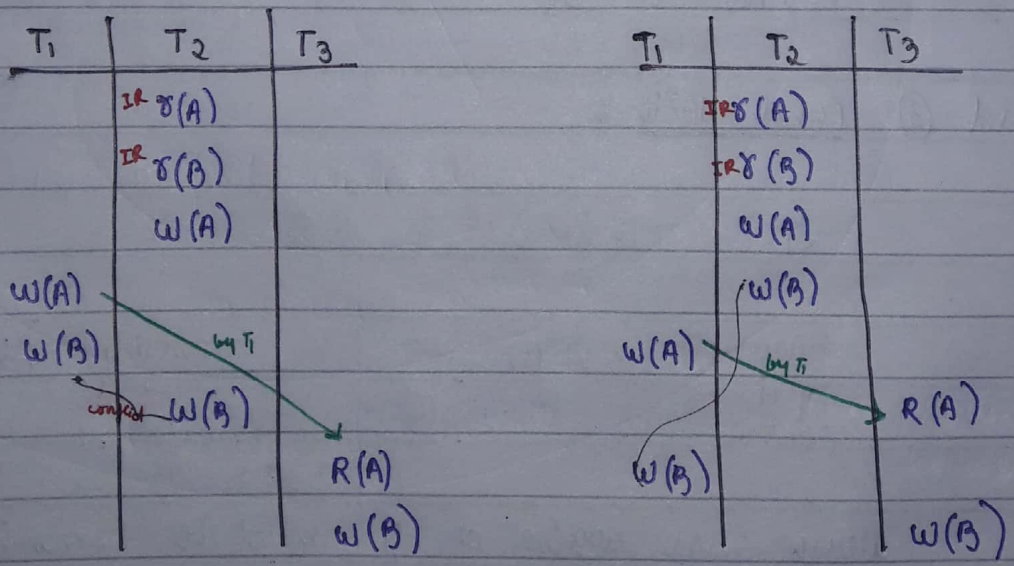
S_1 and S_2 not view equal.
Not conflict equal.

Eg -



∴ Not view Equal, Not conflict equal.

Ques: S_1 :



IR — ✓
UR — ✓
FR — ✓ A by T_1

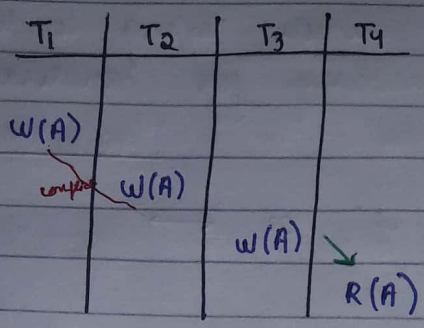
∴ View Equal.
(Not conflict equal)

S_1 : $W_1(A)\ W_2(A)\ W_3(A)\ R_4(A)$

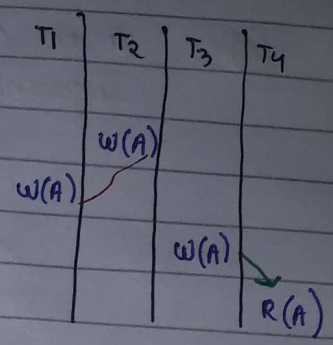
S_2 : $W_2(A)\ W_1(A)\ W_3(A)\ R_4(A)$

Ques:

S_1:



S_2:



IR — Not Possible

UR — ✓

FR — ✓

∴ View Equal

(Not Conflict Equal)

Note

① If S_1 S_2 schedules are conflict equal, then S_1 S_2 also view equal schedule.

② If S_1 S_2 schedules not conflict equal, then S_1 S_2 schedules may or may not view equal

③ If schedule S is conflict serializable schedule, then S also view serializable

④ If schedule S is not conflict serializable schedule, then Schedule S may or may not view serializable.

⑤ Conflict serializable schedule testing condition is only sufficient but not necessary for serializability testing.

↓

if it fails we cannot say that it is not serial may or may not

(i.e only if cond) → implies

Testing cond of conflict Serializ

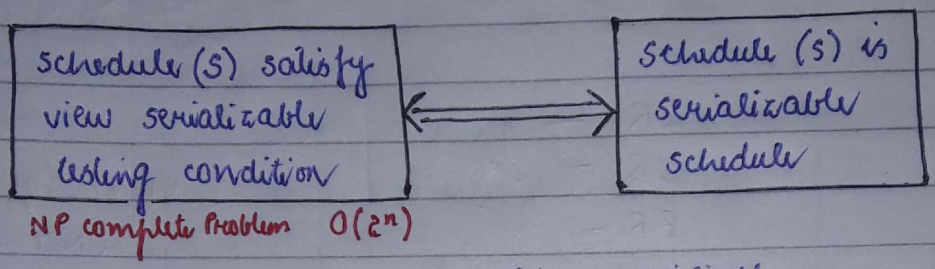
Precedence graph of schedule is acyclic → implies → schedule is Serializable

P. Problem $O(n^2)$

(6) View serializable schedule testing condition is both sufficient and necessary for serializability testing.

↓

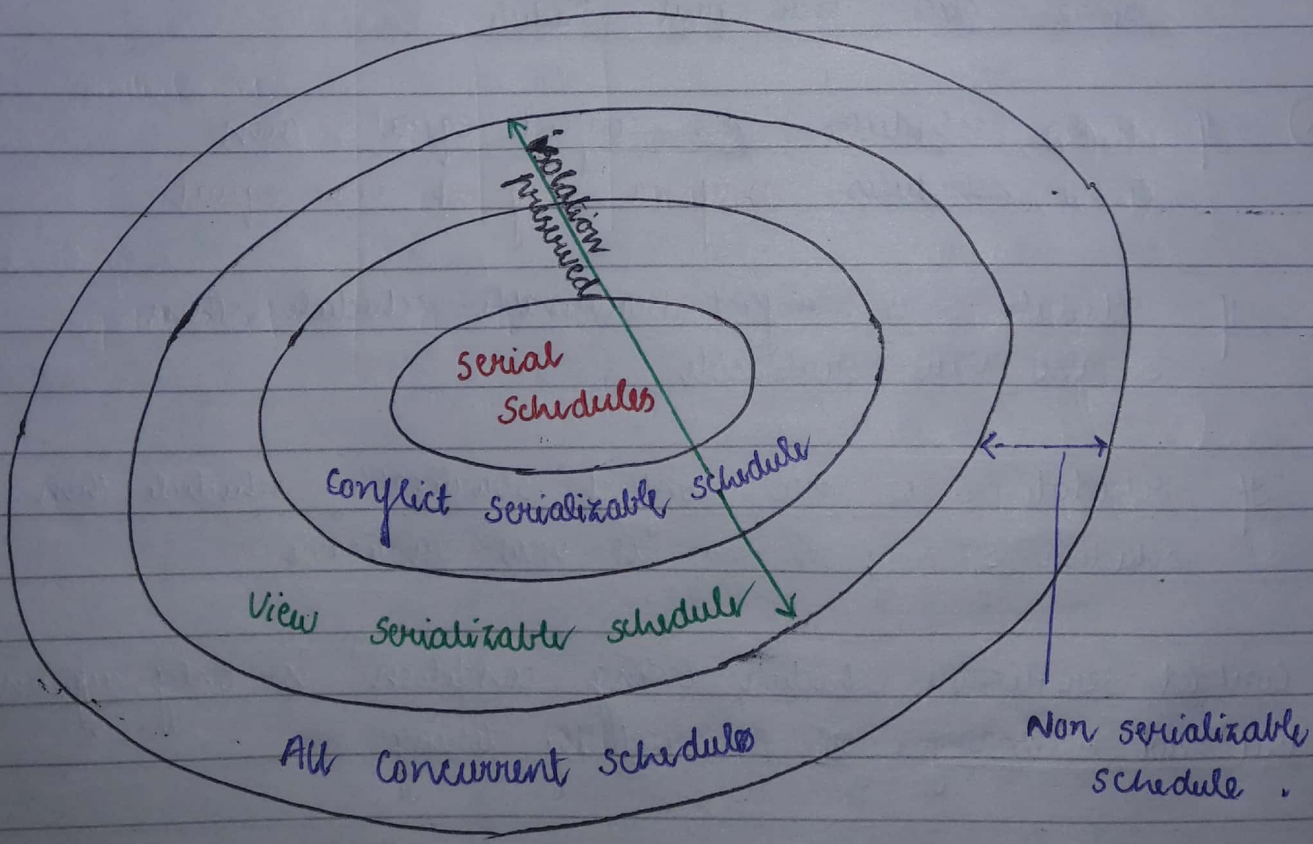
"if and only if" or "⟷"



NP complete Problem $O(2^n)$

if cond satisfied → serializable

cond fails → Not serializable.



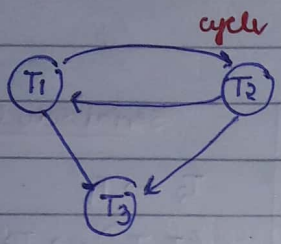
Ques: Test given schedule is view serializable or not

(1) S:

T_1	T_2	T_3
IR $r_1(A)$		
	$W_2(A)$	
$W_1(A)$		
		$W_3(A)$

Solⁿ

a)



∴ Not conflict serializability.

b) For view serializable –

	Given schedule (S)	Serial schedule (S')
Final write	A : T_2 T_1 T_3	(T_1 T_2) Then T_3 any order
initial read		T_1 Then (T_2 T_3)

data item	initial read	write
A	T_1	T_1 T_2 T_3

T_1 must be befor T_2 and T_3

Schedule (S) View serial to T_1, T_2, T_3 serial schedule.

View Serializable

②

T_1	T_2	T_3
	$r(A)$	
	$r(B)$	
	$w(A)$	
$w(A)$		
$w(B)$		
	OR	
	$w(B)$	
		$R(A)$
		$w(B)$

	Given Schedule	Serial Schedule
Final write :	$A = T_2\ T_1$ $B = T_1\ T_2\ T_3$	$T_2 \rightarrow T_1$ $(T_1\ T_2) \rightarrow T_3$ } $T_2\ T_1\ T_3$
Initial read :	data | Initial Read | write A | T_2 | $T_1\ T_2$ B | T_2 | $T_1\ T_2\ T_3$	$T_2 \rightarrow (T_1$ $T_2 \rightarrow (T_1\ T_3)$
update read :	A : $T_1 \rightarrow T_3$ ie $W_1(A) \rightarrow R_3(A)$ (W) (R) and T_2 other transaction also writes A	$T_1 \rightarrow T_3$ and $(T_2 \rightarrow T_1$ or $T_3 \rightarrow T_2)$ ie $T_1 \rightarrow T_3$ should not be ~~T_2~~ in between $T_2\ T_1\ T_3$

View Serializable

and

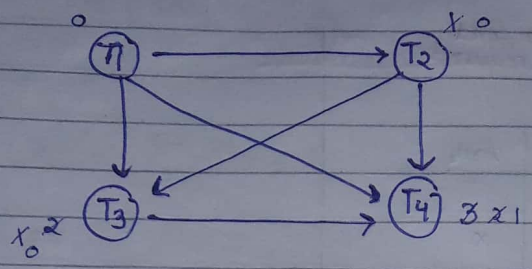
Not Conflict serializable.

(3) S : $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

a) No. of serial schedule conflict equal to sched (S)
b) No. of serial schedule view equal to sched (S)
c) How many serial schedule view equal to serial S but not conflict equal

⇒ 6 - 1
⇒ 5 Ans

a)



T_1 T_2 T_3 T_4

∴ No. of serial schedule conflict equal = 1 Ans

b)

	Given schedule	Serial schedule
Final write:	B : T_1 T_2 T_3 T_4	$(T_1\ T_2\ T_3) \rightarrow T_4$
Initial Read:	data item | Initial R | Write A | T_1 | No one	–

∴ inital read does not matter

Updat read: No one read after write –

∴

_ _ _ T_4

⇒ 3!

⇒ 6 Ans

∴ 6 serial schedules are view equal.

(T1) S_3: $r_1(A)$ $r_3(D)$ $w_1(B)$ $r_2(B)$ $w_3(B)$ $r_4(B)$ $w_2(C)$ $r_5(C)$ $w_4(E)$ $r_5(E)$ $w_5(B)$

Given schedule

Final write:

A = No one

B = T_1 T_3 T_5

C = T_2

E = T_4

} No cond required

Initial read:

data	initial R	writ
A	T_1	x
D	T_3	X

Update read:

Write → Read	Other trans writing data item.
$w_1(B) \rightarrow r_2(B)$	(T_3, T_5) all other writes B
$w_3(B) \rightarrow r_4(B)$	(T_1, T_5)
$w_2(C) \rightarrow r_5(C)$	–
$w_4(E) \rightarrow r_5(E)$	–

View Equal Serial sch(s)

(T_1 T_3) → T_5

$T_1 \xrightarrow{\times\ (T_3\ T_5)\ \text{Not enter R\&W}} T_2$ ✓

$T_3 \xrightarrow{\times\ (T_1, T_5)} T_4$ ✓

$T_2 \longrightarrow T_5$ ✓

$T_4 \longrightarrow T_5$

T_1 T_2 T_3 T_4 T_5

T_3 T_4 T_1 T_2 T_5

2 Ans

Classification based on Recoverability –

Concurrent execution may leads :

1. Irrecoverable schedule
2. Cascading Rollback problem
3. Lost update problem

} These problem can occur even if schedule is 'serializable'

Irrecoverable Schedule –

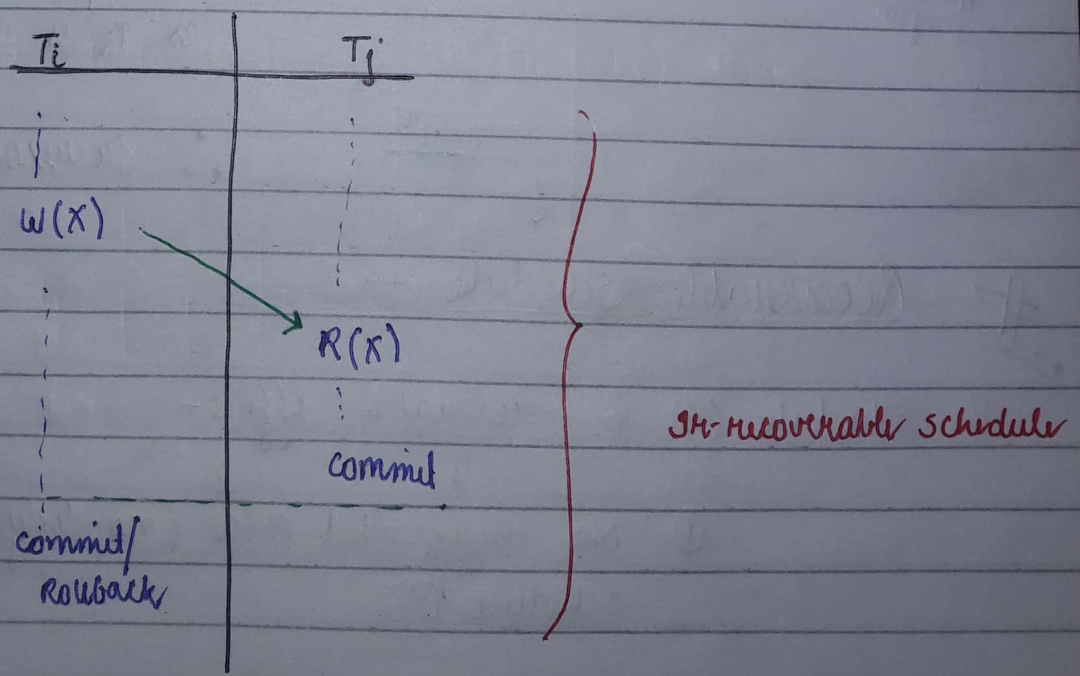
↓

Unable to recover or rollback

⇓

(Rollback of committed transaction)

Schedule (S) is irrecoverable iff Transaction T_j reads data item 'X', which is updated by T_i and, commit of T_j before commit / rollback of T_i.

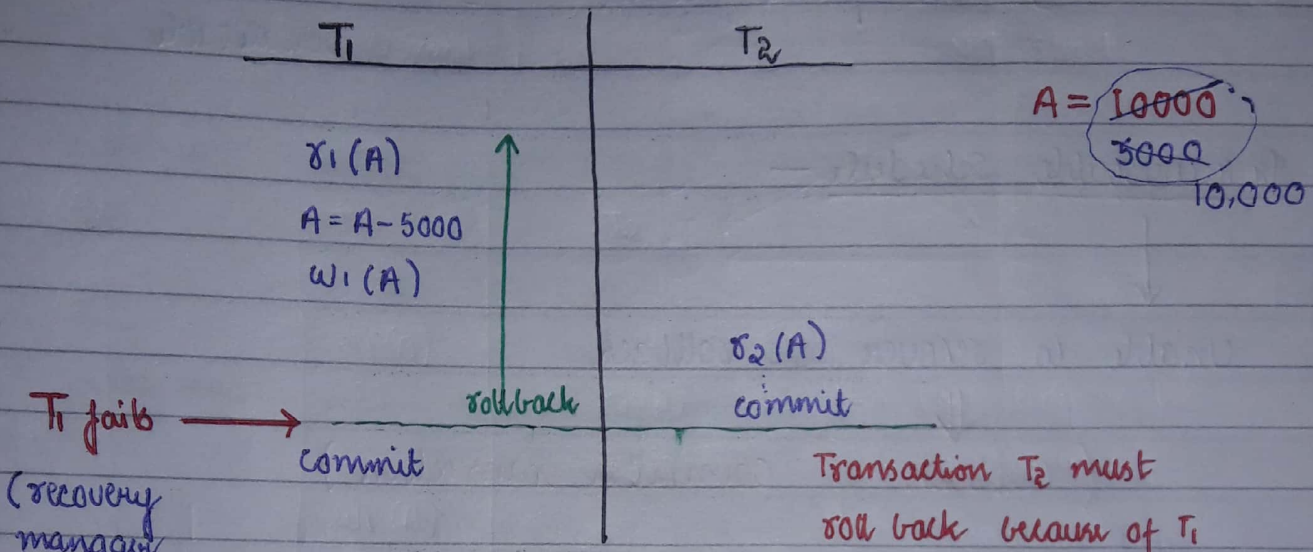


eg -

T_1 : Withdraw 5000 from 'A' [$r_1(A)$, $A = A - 5000$, $w_1(A)$, commit]

T_2 : Check balance of 'A' [$r_2(A)$, commit]

Let T_1 and T_2 executed concurrently.



Solⁿ

∴ It is recoverable

Recoverable Schedule —

Schedule (S) is recoverable iff :-

① No uncommitted read (no dirty read) in schedule (S)

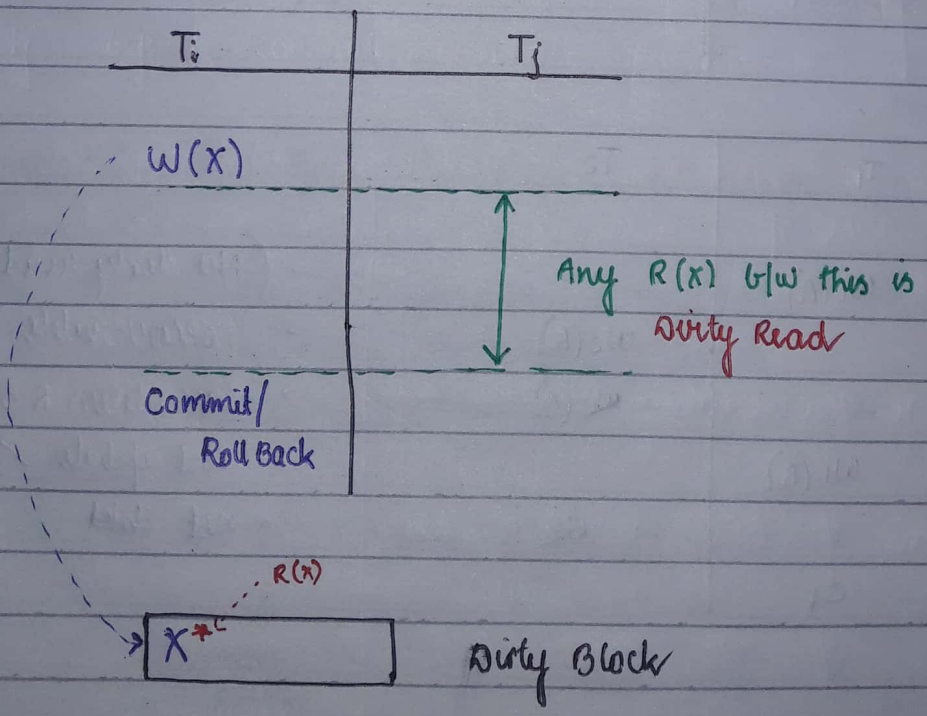
[OR]

② If Transaction T_j reads data item 'X' which is updated by transaction T_i, then commit of T_j must be delayed until commit / rollback of T_i.

	T_i	T_j
1.	W(X)	
2.		R(X) ← dirty read
3.	Commit/RollBak	
4.		Commit

* **Uncommitted read (Dirty Read)** —

Transaction T_j reads data item 'X' which is updated by uncommitted transaction T_i



Ques: Test which of the schedule is recoverable –

①

T_1	T_2
W(A)	
	R(A)
	rollback
commit	

- (dirty read exist)
- Recoverable
- Not cascadeless roll back
- Not strict

②

T_1	T_2
$r_1(A)$	
	$r_2(A)$
	$w_2(A)$
$w_1(A)$	
	C2
C_1	

- (NO dirty read)
- recoverable
- Cascadeless R·B
- lost update ✗
- Not strict

③

T_1	T_2
$w_1(A)$	
	$w_2(A)$
	$w_2(B)$
$w_1(B)$	
	C_2
C_1	

- (NO dirty read)
- recoverable.
- Cascadeless R·B
- lost update ✓
- not strict

④

T_1	T_2
$w(A)$	
	$r(A)$
	$r(B)$
$w(B)$	
C_1	
	C_2

- (Dirty read exist)
- Recoverable.
- (C_2 after C_1)
- Not cascadeless R.B
- lost update ✗
- Not strict.

⑤

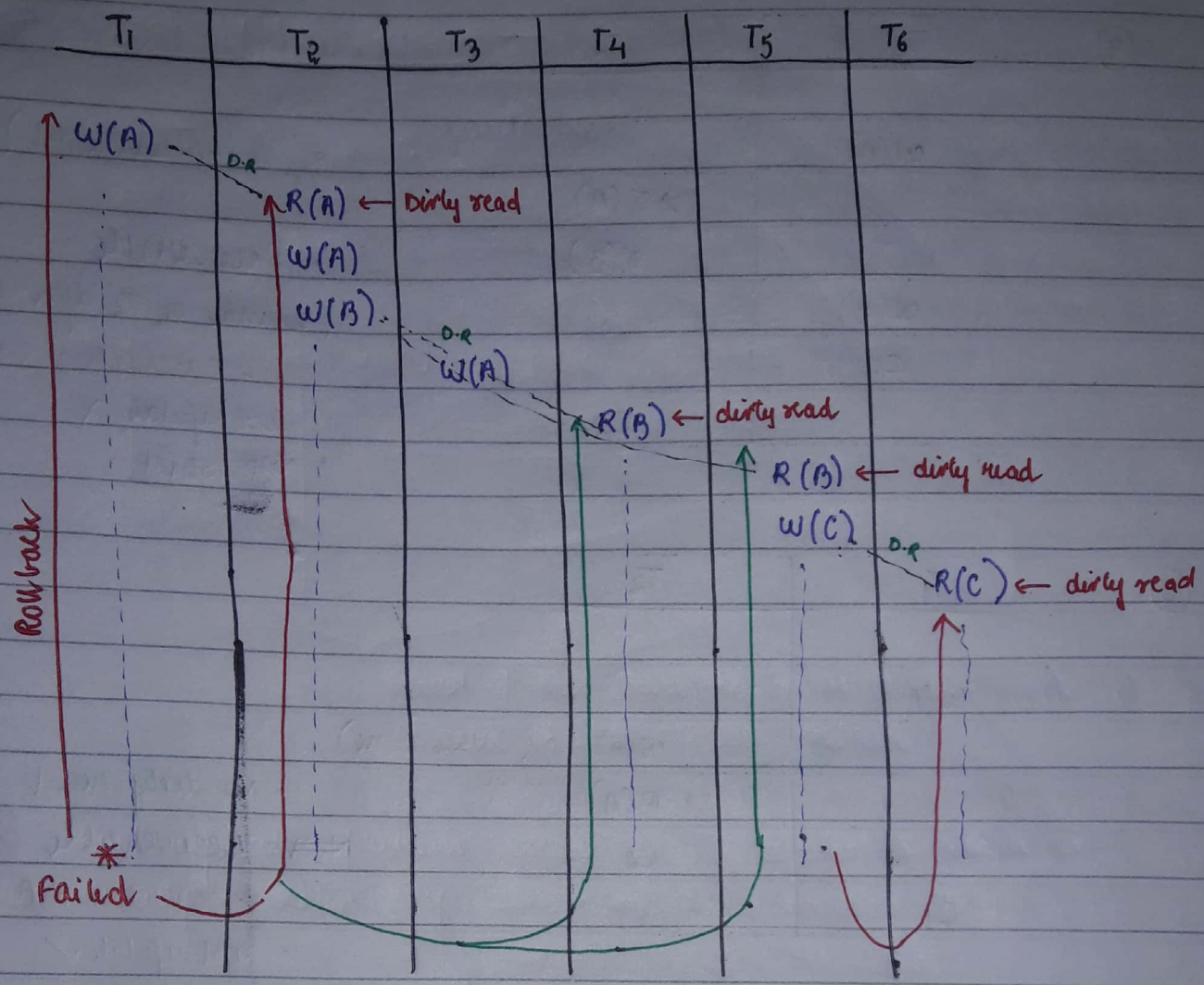
T_1	T_2
$w(A)$	
	$w(A)$ (write done)
	$R(A)$
	C_2
C_1	

not by this → $w(A)$; read this → $R(A)$

- No dirty read
- ~~Non~~ Recoverable.
- Cascadeless R.B
- lost update ✓
- Not strict.

Cascading RollBack Schedule (Problem)

Because of failure of some transaction, many other set of transaction are forced to rollback.



Because of failure of T_1, forced to rollback T_2, T_4, T_5, T_6

Disadvantage of cascading rollback —

① wastage of CPU execution time.

② wastage of I/O access cost.

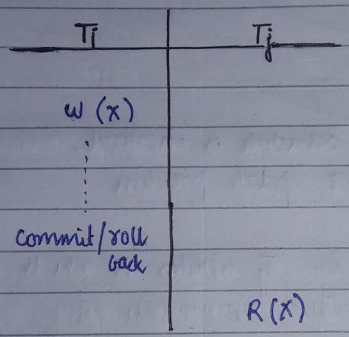
⇓ soln

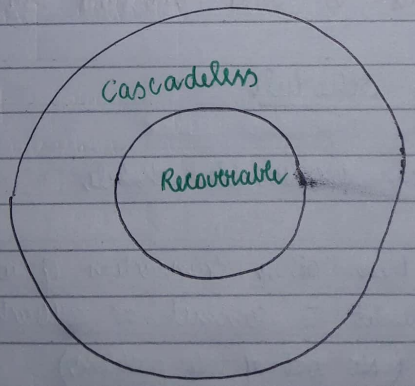
Cascadeless Rollback Recoverable Schedule

Cascadeless Rollback Recoverable Schedule —

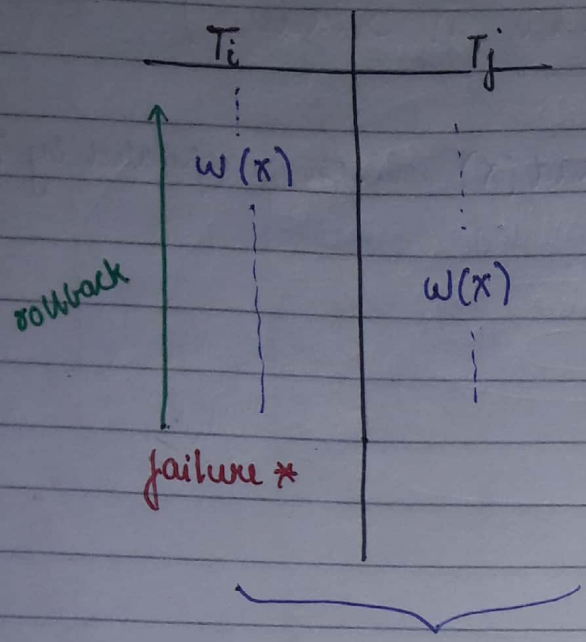
(No Dirty Read in Schedule)

- Transaction T_j should delay read(X) which is updated by T_i, until T_i commit or rollback.
- Dirty reads are not allowed





Lost Update Problem —



update of T_j is lost because of failure of T_i

x : 100
200 ← written by T_i
50 ← written by T_j after failure
roll backs to 100

Result of the schedule is incorrect, because of lost update problem.

- lost update problem occur if T_j writes 'x' which is already written by uncommitted transaction T_i

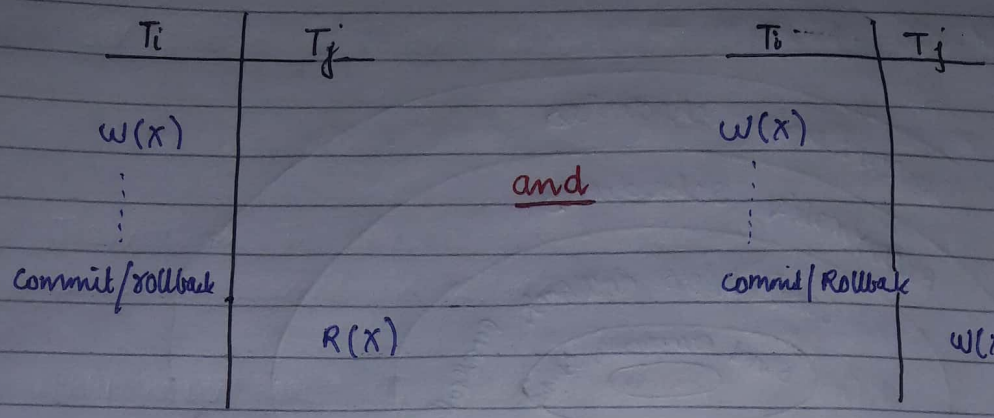
⇓

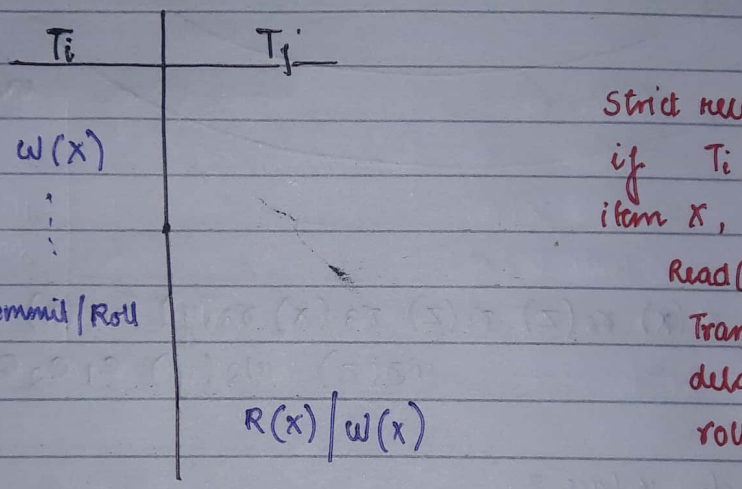
Solⁿ is Strict Recoverable schedule

Strict Recoverable schedule —

① Cascadeless rollback recoverable schedule

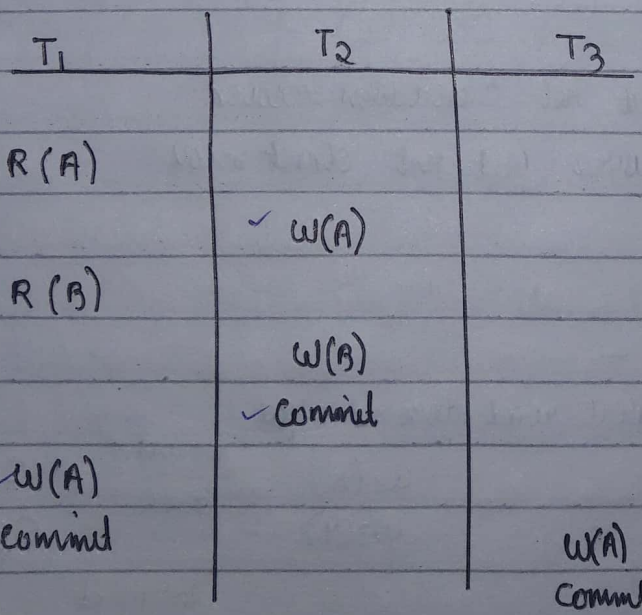
and ② If T_i writes 'x' then other transaction T_j write of 'x' must be delayed until T_i commit or rollback.
(No lost update problem)



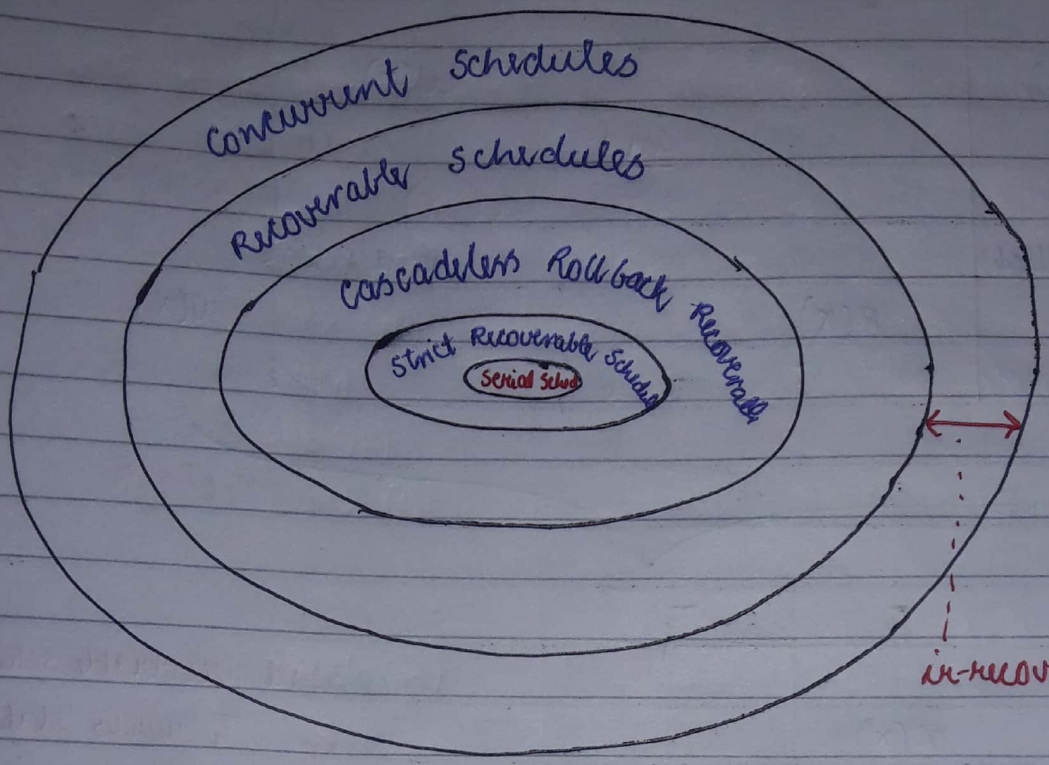


Strict recoverable schedule
if T_i writes data item X, then
Read(x) / write(x) of
Transaction T_j must
delay until commit/
rollback of T_i

Eg -



⇒ Strict Recoverable



Ques: $S: r_1(x)\ r_2(z)\ r_1(z)\ r_3(x)\ r_3(y)\ w_1(x)\ w_3(y)\ r_2(y)\ w_2(z)\ w_2(y)\ c_1\ c_2\ c_3$

which is true ?

a) unrecoverable
b) recoverable but not cascadeless recover
c) cascadeless recover but not strict recov
d) strict recov

check for strict rec –

initial read does not affect

$w_1(x)$ – (no read x) – c_1 ✓

$w_3(y)$ – ($r_2(y)$) – c_3 ✗

dirty read

(Also violation for cascadeless)

∴ Not cascadeless

Now for recoverable, it should be.

$w_3(y) \ldots\ldots r_2(y) \ldots\ldots C_3\ C_2$

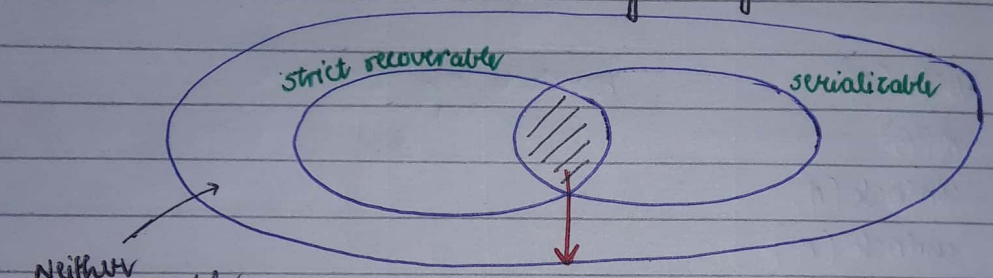
But
C_2 commit before C_3
∴ Not recoverable

∴ Irrecoverable Ans

Note —

① Strict recoverable schedule may or may not be serializable.

② Serializable schedule may / may not be recoverable.



Schedule (S) is correct schedule iff
S is strict recoverable and
S is serializable.

15/11/17

Concurrency Control Protocol —

Concurrency Control Protocol should not allow to execute

① Non serializable schedule (violate isolation)

② Non strict recoverable schedule (violate Atomicity & Durability)

Locking Protocol –

Lock is a variable used to identify the status of data item.

```
Trans (T1)
Lock (A)        : granted by concurrency controller.
R(A)
W(A)
Lock (B)        : denied by concurrency controller.
 :
 :
Lock (B)        : granted by C.C.
R(B)
unlock (A)
unlock (B)
```

Types of Lock –

1. Shared lock (S) → read only lock
2. Exclusive lock (X) → read/write lock.

Eg –

```
Trans (T1)
S(A)        : granted shared lock
R(A)        } only read
X(B)        : granted exclusive lock.
R(B)        }
W(B)        } read and write
```

Lock compatable Table —

Holded by T_i

data item "A"	S	X
Requested by T_j : S	Yes	NO
Requested by T_j : X	NO	NO

T_1	T_2	
S(A)		
	S(A)	: granted
R(A)		
	R(A)	
S(B)		
	X(B)	: denied
R(B)		

Eg -

T_1	T_2
W(A)	
	R(A)
	R(B)
W(B)	
	C_2 ✓
C_1 ✓	

Applying view serializable.

update read: $W_1(A) \rightarrow R_2(A)$: $T_1 \rightarrow T_2$

Initial read: $B = T_2$ with T_1 : $T_2 \rightarrow T_1$

serial schedule Not possible

∴ Non serializable
Non recoverable

So, we need to place locks in some order.

T_1	T_2
X(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
	S(B)
	R(B)
	U(B)
X(B)	
W(B)	
U(B)	

U: unlock.

∴ shared / exclusive locks not sufficient to avoid non serializable and non strict recoverable.

2 Phase locking Protocol (2PL) —

(guaranteed serializability)

- Transaction (T) allowed to request lock on any data item in any mode (S/X) until first unlock of - Trans(T)
- All locks together (i.e 1st) and unlock together i.e later

↓ growing phase ↓ shrinking phase.

Trans (T)

S (A)
X (B)
S (C)
X (D)
U (C)
U (D)
U (A)
U (B)

growing phase (locking phase)

1st unlock of T → U (C) *

locking point of trans (T)

shrinking phase (unlocking phase)

lock point — Position of last lock / 1st unlock of transaction T.

T_1 : Transfer 500 from A to B

T_2 : display total balance of AB

T_3 : set A, B as 2000

Trans (T_1)

X (A)
R (A)
W (A)
X (B)
U (A)
R (B)
W (B)
U (B)
commit

Satisfy 2PL

Trans (T_2)

S (A)
R (A)
S (B)
U (A)
R (B)
U (B)
disp (A+B)
commit

Satisfy 2PL

Trans (T_3)

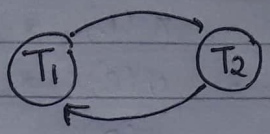
X (A)
W (A)
X (B)
U (A)
W (B)
U (B)
commit

Satisfy 2PL

∴ Execute concurrently : Concurrent execution of T_1 T_2 T_3 (guaranteed serializable)

Eg —

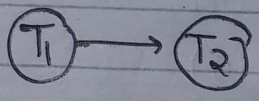
T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
U(A)	
	S(A)
	R(A)
	S(B) ← denied till T_1 unlocks B
	R(B)
R(B)	
W(B)	
U(B)	
Commit	



∴ Not Allowed by 2PL

⇓

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
U(A)	
	S(A)
	R(A)
R(B)	
W(B)	
U(B)	
	S(B)
	R(B)



Conflict serializable S

Allowed by 2PL

Note

- If schedule executed by 2PL, then schedule gaurenteed is conflict serializable schedule.
- Conflict equal schedule is based on lock points order of the transaction.

S :

T_1	T_2	T_3
⋮	⋮	⋮
*	*	
		*

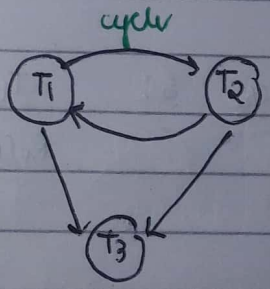
Schedules executed by 2PL
↓
guranteed conflict serializable schedule
∴ Equal serial schedule
$T_2 \rightarrow T_1 \rightarrow T_3$
(order of lock point)

- If schedule is not conflict serializable schedule (cyclic precedence graph schedule), then schedule not allowed to execute by 2PL

S :

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

⇒



Not conflict serializable

checking view serializable,

$T_1 \rightarrow (T_2\ T_3)$

$(T_1\ T_2) \rightarrow T_3$

$T_1\ T_2\ T_3$ View serializable

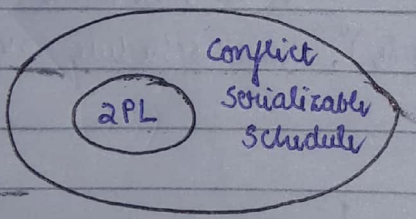
Writing using locks —

T_1	T_2	T_3
X(A)		
R(A)		
	X(A) ← denied	
	W(A)	
W(A)		
U(A)		
		W(A)

∴ Above schedule is not conflict seriable but view seriable

∴ "Not allowed by 2PL"

- Every schedule which is allowed by 2PL is always conflict serializable, but not every conflict serializable schedule is allowed by 2PL



Ques: S:

T_1	T_2	T_3
$R_1(A)$		
	$W_2(A)$	
	$W_2(B)$	
		$R_3(B)$
		$R_3(C)$
$W_1(C)$		

Cascadless Recov X

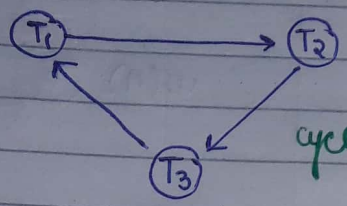
strict " X

(May or may not recoverable)

which is true ?

Schedule (S) is

(a) Conflict SS and allowed by 2PL
(b) Conflict SS but not allowed by 2PL
(c) not conflict SS and ,,
(d) ,, and view serializable.



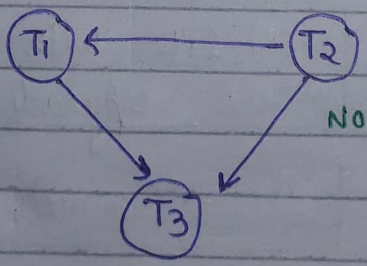
cycle ∴ Not conflict serializable
⇒ Not allowed by 2PL

Ques:

T_1	T_2	T_3
	x(A) W(A) x(B) U(A)	
x(A) W(A) x(B) U(A)		
	W(B) U(B) U(A)	
* x(B) W(B) U(B)		x(A) W(A)
		x(A) * W(B) U(B)

✓ Recoverable
✓ Cascade Recov
✗ Strict "

which is true, above options.



No cycle

∴ Conflict serializable

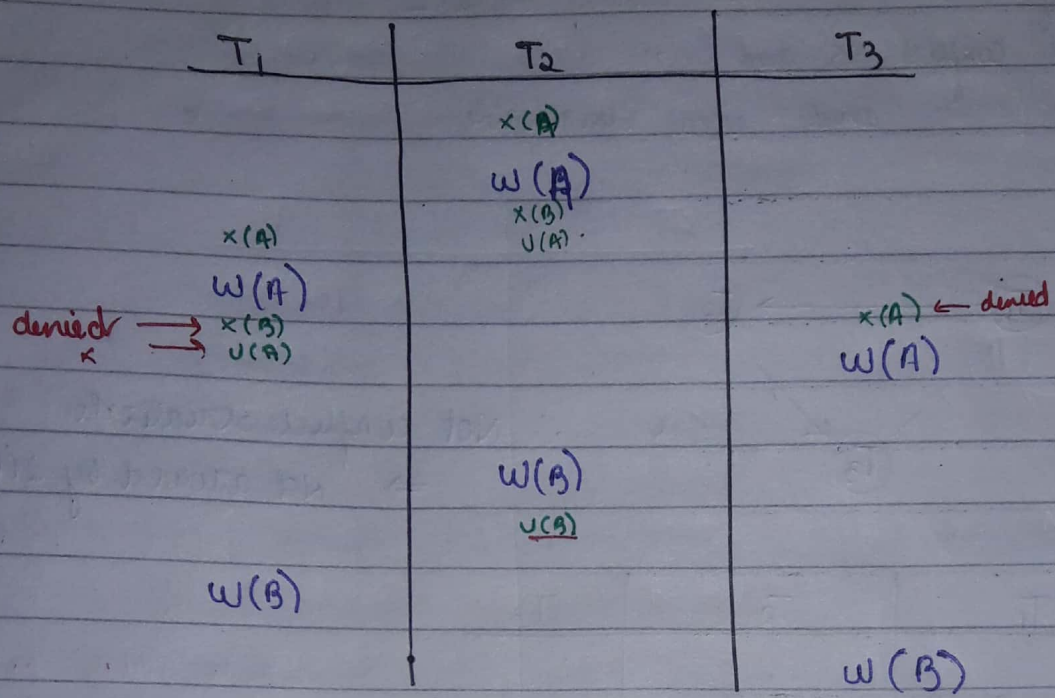
View

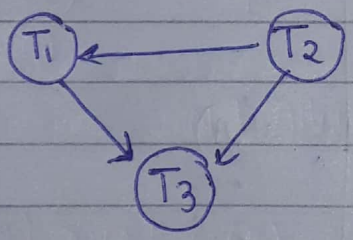
Allowed by 2PL.

∴ Equal serial schedule
$T_2 \rightarrow T_1 \rightarrow T_3$

(a)

Ques: $w_2(A)\ w_1(A)\ w_3(A)\ w_2(B)\ w_1(B)\ w_3(B)$





conflict serializable.

Not allowed by 2PL (from above fig)

(6)

Lock Upgrading 2PL (vs) without lock upgrade 2PL –

2PL without lock upgrade	2PL with lock upgrade (default 2PL)

S: | T_1 |
|---|
| X(A) |
| R(A) |
| ⋮ |
| W(A) |
| U(A) |

↑ Even if no change in value of A, NO other transaction allowed to use data item 'A' in shared mode ↓

S: | T_1 |
|---|
| S(A) |
| R(A) |
| ⋮ |
| X(A) |
| W(A) |

upgrade

↑ Other transaction can use data item 'A' in shared mode ↓

Lock upgrade of transaction must complete in growing phase.

Less degree of concurrency

T_1	T_2
X(A)	
R(A)	
	S(A) ← denied
	R(A)
W(A)	
U(A)	

Not allowed by 2PL without lock upgrade (even when it is serializable)

More degree of concurrency

T_1	T_2
S(A)	
R(A)	
	S(A) ← grants
	R(A)
	U(A)
X(A)	
W(A)	
U(A)	

Allowed to execute with lock upgrade.

Ques: Test the given schedule allowed by 2PL or not ?

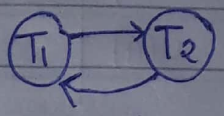
① S: $r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ r_1(B)\ w_1(B)$

Soln:

T_1	T_2
X(A)	
R(A)	
W(A)	
~~X(A)~~ S(B)	
U(A)	
	S(A)
	R(A)
	~~U(A)~~
	S(B) ← denied
	R(B)
	U(B)
	U(A)
R(B)	
X(B)	
W(B)	
U(B)	

growing

(Not allowed during Shrinking phase)



cyclic

Not Conflict

↓

Not 2PL.

Not Allowed by 2PL

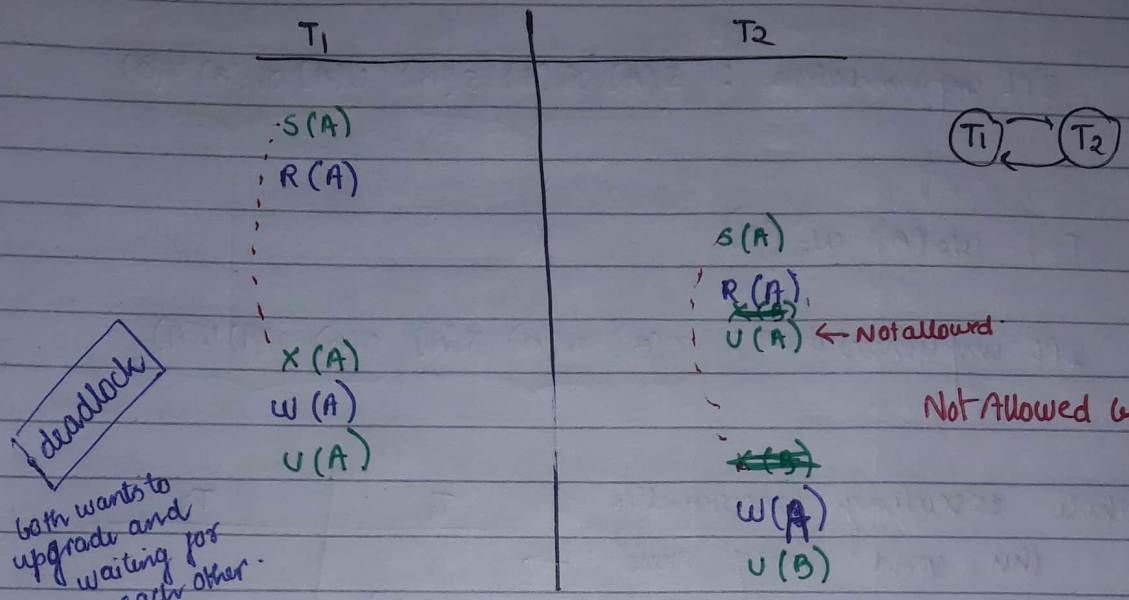
② S: $r_1(A)\ r_2(A)\ r_2(B)\ w_1(B)\ w_1(A)$

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	S(B)
	R(B)
	U(A)
	U(B)
X(B)	
W(B)	
W(A)	
U(B)	
U(A)	

upgrade

Allowed by 2PL

③ S: $r_1(A)\ r_2(A)\ w_1(A)\ w_2(A)$



Limitations of 2PL —

① 2PL restriction may leads to deadlock.

② 2PL restriction may lead to starvation

③ 2PL condition not sufficient to avoid:

a) IR-recoverable schedules

b) Cascading rollback problem

c) ~~Strict recoverable problem~~ Lost update problem

Eg — 2PL restriction may leads deadlock

T_1 : $\gamma_1(A)$ $\gamma_1(B)$

2PL implementation : $S(A)$ $\gamma_1(A)$ $S(B)$ | $U(A)$ $\gamma_1(B)$ $U(B)$

T_2 : $w_2(B)$ $w_2(B)$

2PL implementation : $X(B)$ $w(B)$ $X(A)$ | $U(B)$ $w(A)$ $U(A)$

while executing concurrently this cond may arise →

T_1	T_2
$\gamma_1(A)$	
	$w_2(B)$
$\gamma_1(B)$	
	$w_2(A)$

⇓ 2PL implementation

T_1	T_2
$S(A)$	
$\gamma_1(A)$	
	$X(B)$
	$w_2(A)$
denied → $S(B)$	
$\gamma_1(B)$	
	$X(A)$ ← denied
	$w_2(B)$

Forms Deadlock

Ques: data items A_1 A_2 A_3 ---- A_n

Protocol —

① Transaction should lock all required data item in proper sequence of $A_1, A_2, \ldots A_n$

② Transaction should perform read/write's

③ Transaction unlock's data.

Which is true ?

a) Protocol guaranteed serializable and No deadlock

b) Guaranteed serializable but may cause deadlock

c) Not guaranteed serializable and free from deadlock.

d) None.

T_1	T_2
S(A)	
S(B)	
X(A)	
X(B)	

1st step — lock all the data item

2nd step — Perform R/W

3rd step — Unlock all the data item

It is 2PL

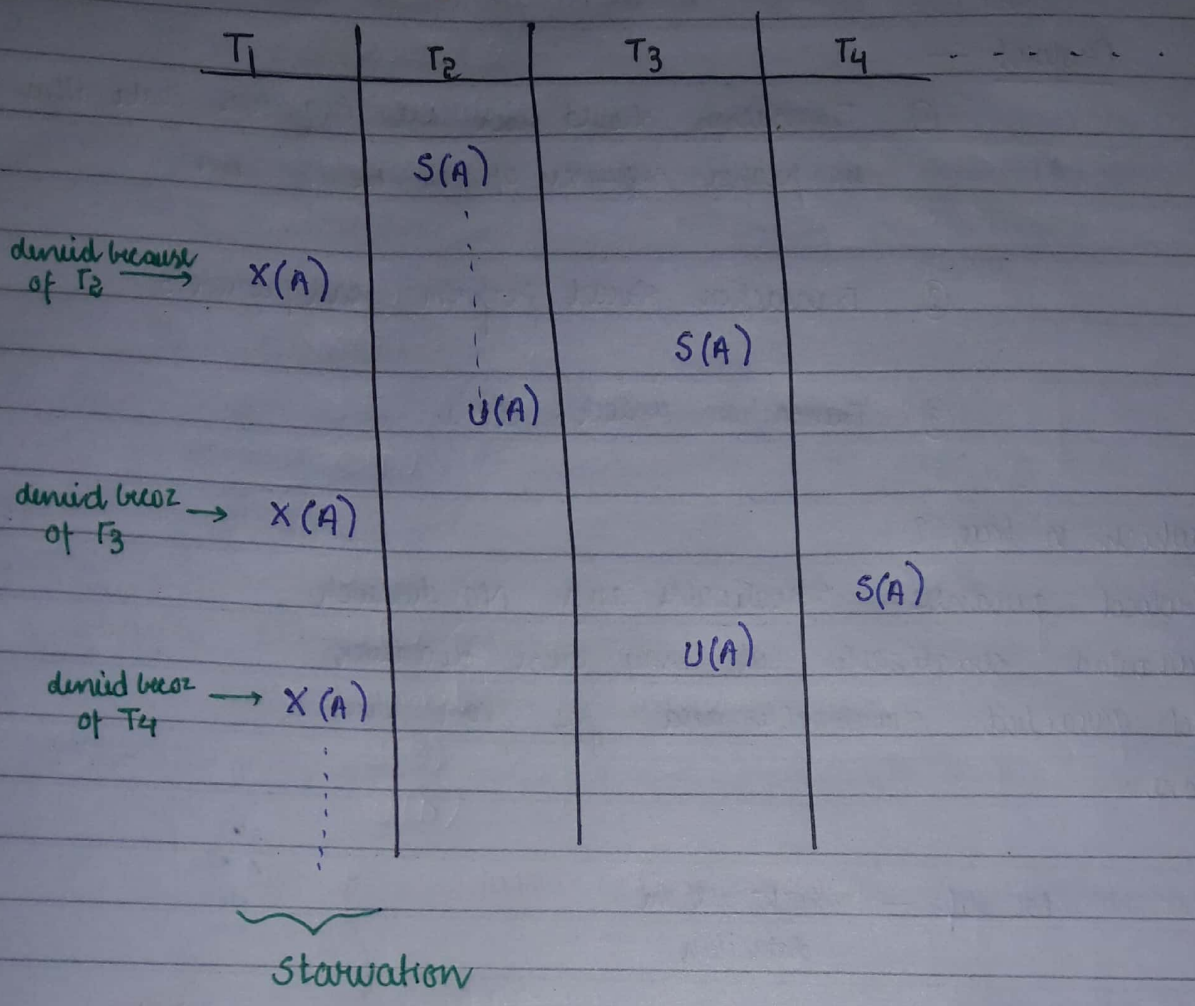
∴ Guaranteed Serializable.

Since, it lock all the data needed in begning.

	T_1	T_2
in start it locks → A_2, A_3, A_6		A_4, A_6, A_2 ← cannot
	X(A_2)	W(A_4)
∴ this executes	W(A_3)	W(A_6)
	W(A_6)	X(A_2)

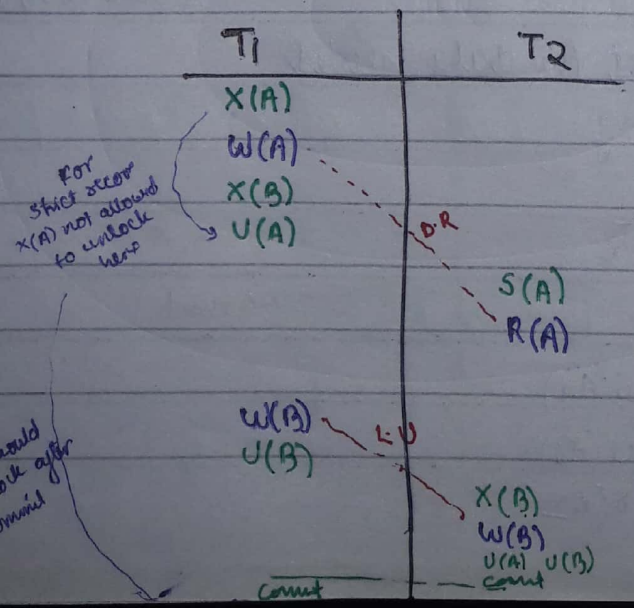
∴ No deadlock.

Eg - 2PL may form starvation.



Eg - 2PL restriction not sufficient to avoid

- Irrecoverable
- Cascading Roll back
- lost update Problem.



→ Irrecoverable schedule
→ Cascading RB possible
→ lost update problem possible

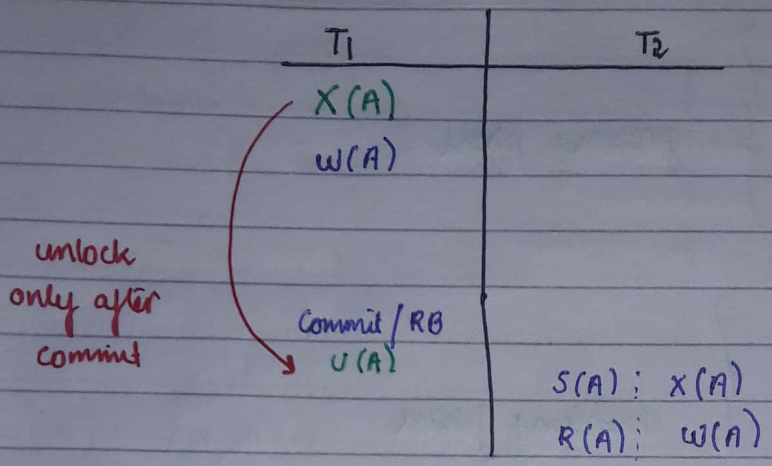
Allowed to execute by 2PL

⇓

Conflict serializable

Strict Recoverable condition using lock —

All exclusive locks of transaction (T) should hold until the commit/rollback of trans (T)



⇓ modified 2PL protocol.

Strict 2PL Protocol —

- **Basic 2PL** : Lock request of trans (T) not allowed in shrinking phase of trans T.

and

- **Strict Recoverable** : All exclusive locks of transaction T must hold until commit/rollback of trans T.

→ Strict 2PL protocol guarantees —

1. Serializable
2. Strict Recoverable-

- It is also not free from —
  1. Deadlock
  2. Starvation

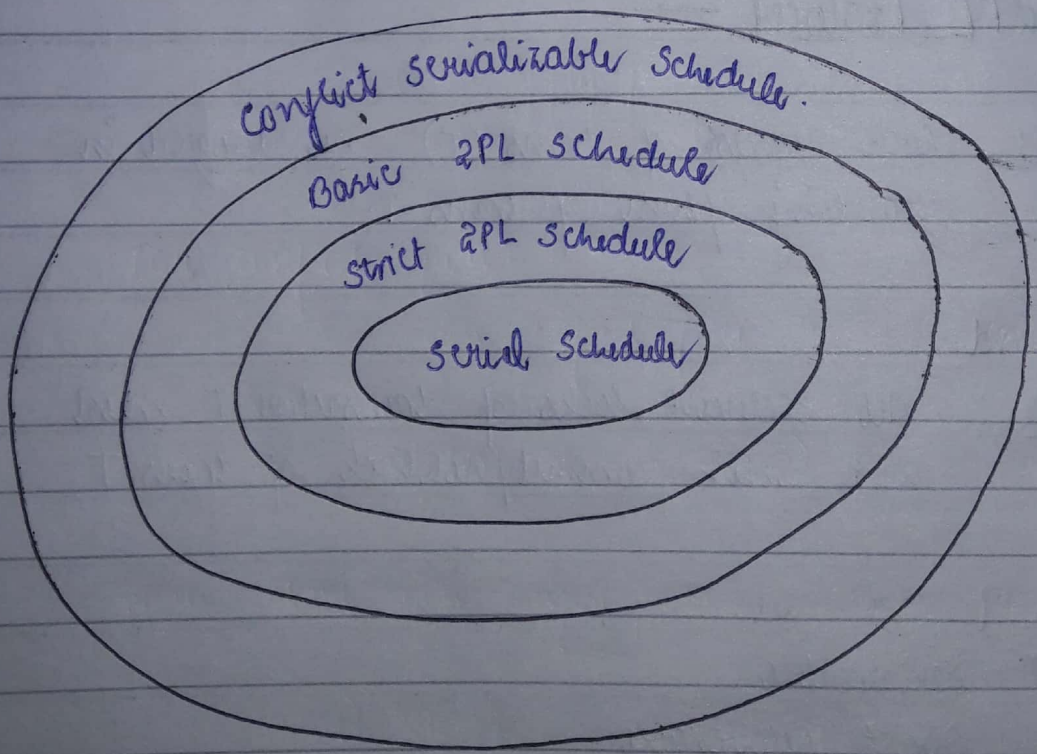
Trans (T)

Strict 2PL transaction

S(A)
X(B)
S(C)
X(D)
⋮
U(A)
U(C)
commit
U(B)
U(D)

growing phase

shrinking phase.



Pg-61

(23) Consider the transaction.

T_1 : $r_1(A)$ $r_1(B)$ $w_1(B)$

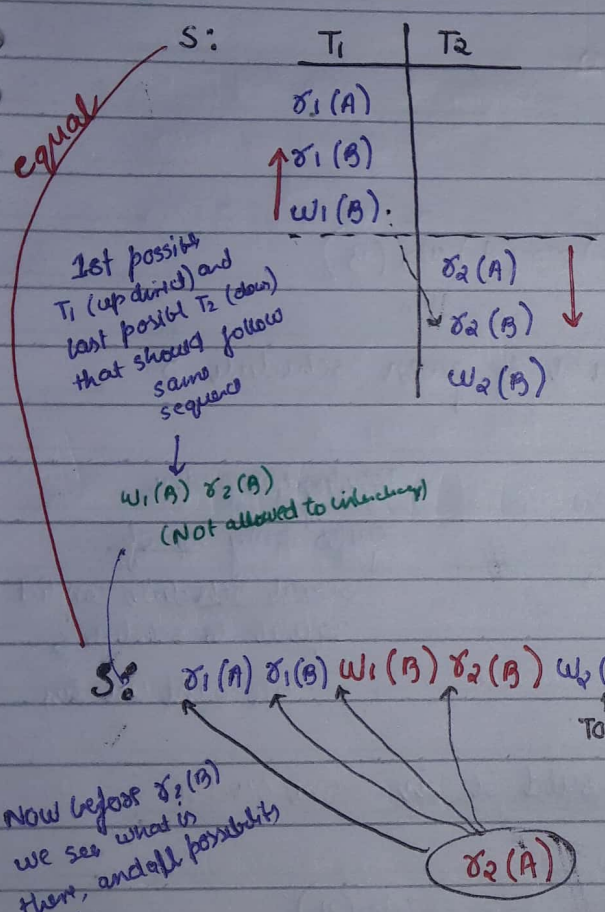
T_2 : $r_2(A)$ $r_2(B)$ $w_2(B)$

> Total concurrent schedule = ${}^6C_3$ = 220

How many non serial schedules b/w T_1 and T_2 are serializable.

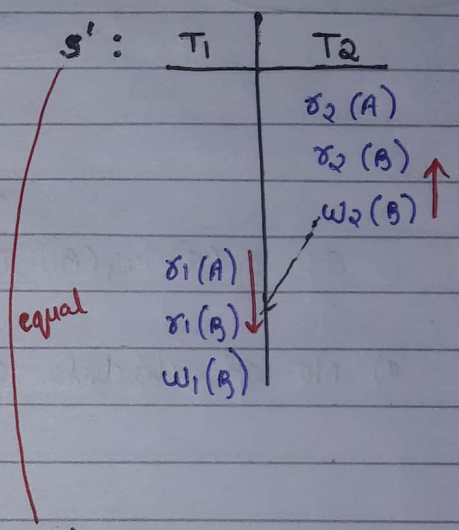
Soln:

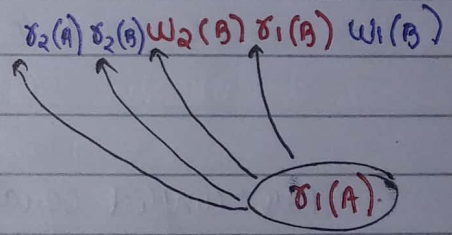
Concurrent execution of T_1, T_2 equal to $T_1 \rightarrow T_2$ serial



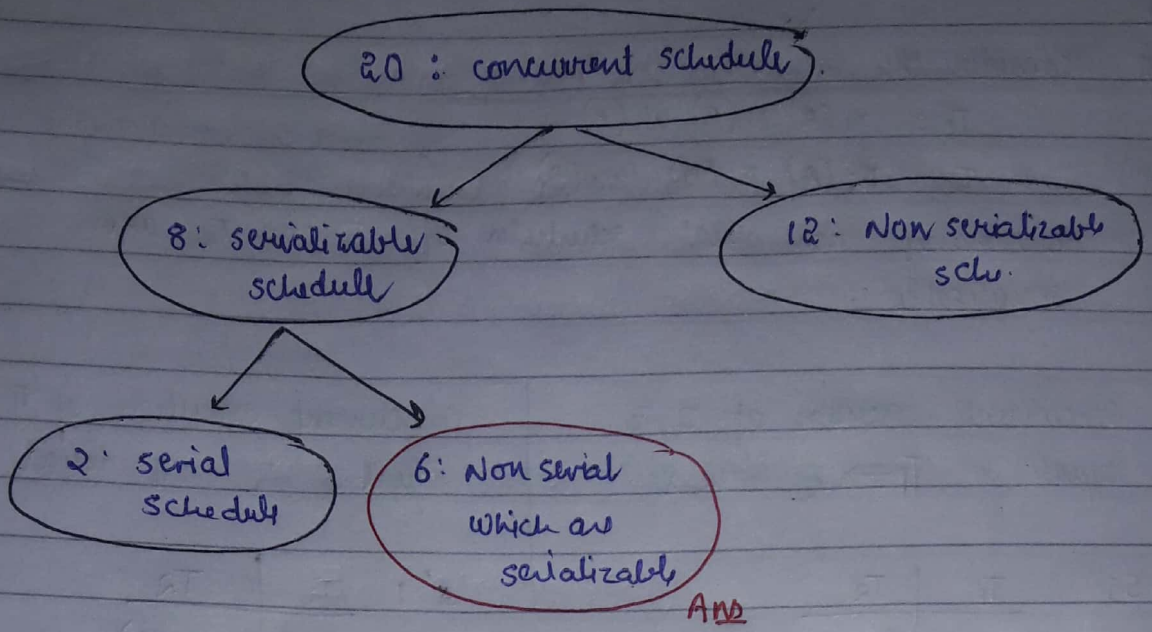
∴ 4 serializable schedule

Concurrent execution of T_1 T_2 equal to $T_2 \rightarrow T_1$ serial



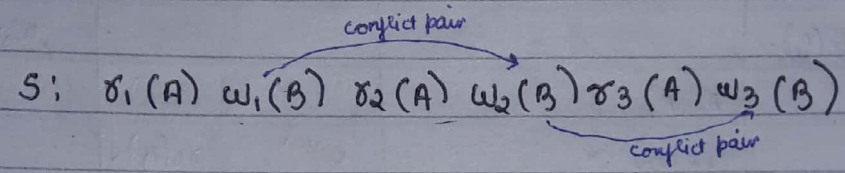


4 serializable schedule



Pg 61

(27)



a) No. of schedule conflict equal to given schedule S

So we need to apply permutation comb

Topological order gives only no. of serial schedule conflict equal to schedule S
∴ we can't use this

For conflict equal,
the conflict pair should be in same order -

$w_1(B)$ ..... $w_2(B)$ ..... $w_3(B)$

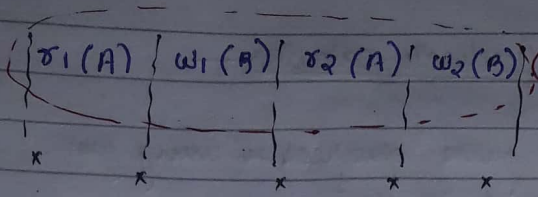
same order.

Now, start with longest possible relation and place other.

$r_1(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ } 3 schedule.

For $r_2(A)$ ✓ ✓ ✓

can be placed anywhere before $w_3(B)$



can be placed anywhere before w_3 as it does not form conflict pair with any. → 5 schedule

$\therefore$ Total = 3 × 5

= 15 schedule Ans

b) No. of schedule view equal to schedule (S).

S : $r_1(A)\ w_1(B)\ r_2(A)\ w_2(B)\ r_3(A)\ w_3(B)$

Here, No violation of initial read, as no one is writing it
No read update.

Checking for final write.

B : $T_1\ T_2\ (T_3)$ $(T_1\ T_2) \rightarrow T_3$

$\therefore$ 2 options.

From previous: $w_1(B)\ w_2(B)\ w_3(B)$ } 15

$w_2(B)\ w_1(B)\ w_3(B)$ } 15

30

Ans

H/W Ques: S: $w_1(A)\ w_1(B)\ r_2(A)\ w_2(B)\ r_3(A)\ w_3(B)$

a) Find No. of conflict equal

b) " " " View Equal.

TRC (Tuple Relational Logic)

First Order logic and Predicate calculus —

Ques: Write FOL statement for given specification using fun

$C(x)$: x student in class

$S(x)$: x studied maths.

① Some student in class studied maths

$$\exists x \, (C(x) \wedge S(x))$$

Some 'x' such that
x student in class
and
x studied maths.

② Every student in class studied maths.

$$\forall x \, (C(x) \rightarrow S(x))$$

For every x
if x student in class, then
x must study maths.

Wrong statement → $\forall x \, (C(x) \wedge S(x))$

every x must be student in class.

$X = \{ A \;\; B \;\; C \;\; | \;\; D \;\; E \}$

A B C : student in class, studied maths.

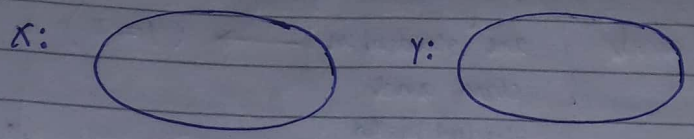
D E : Not stud in class.

$\forall x (C(x) \rightarrow S(x))$ — T

$\forall x (C(x) \wedge S(x))$ — F

③ Atleast 2 student in class studied maths.

2, or more.



Some X student in class and studied maths

Some Y student in class and studied maths

and $X \neq Y$.

$\Rightarrow$ All — Almost I

$\exists x (C(x_1) \wedge S(x_1) \wedge C(x_2) \wedge S(x_2) \wedge x_1 \neq x_2)$

$$\exists X \left(C(X) \wedge S(X) \wedge \exists Y \left(C(Y) \wedge S(Y) \wedge (X \neq Y) \right) \right)$$

$$\Rightarrow \exists x \exists y \left(C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y) \right)$$ Ans

④ Only one student in class studied maths.

Only one = (Atleast one student in class studied maths — atleast two stud in class stud. maths)

= atleast one but not atleast 2

$P - Q \equiv P \wedge \neg Q$ (atleast one, atleast 2)

$= P \wedge \neg Q$

$\Rightarrow \exists x (C(x) \wedge S(x)) \wedge \neg (\exists x \exists y (C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y)))$

OR

Method 2:

{ some x student in class and studied maths } and (every y who are student in class and studied maths) ⟶ (Y = X) }.

$$\exists x \left(C(x) \wedge S(x) \wedge \forall y \left(\left(C(y) \wedge S(y) \right) \longrightarrow (y = x) \right) \right)$$

TRC -

TRC query format:

- $\{ T \mid P(T) \}$

T : Tuple variable
P(T) : Formula over tuple variable

↓

Results set of tuple (T) those satisfy formula P(T)

$\exists T_1 \in Stud\ (P(T_1))$
$\forall T_2 \in course\ (P(T_2))$

T_1 T_2 are "Bounded tuple variable"

→ Tuple variable used for result of TRC query must be free & tuple variable.

Ques:

Stud (sid, age)
Course (cid, instructor)
Enroll (sid, cid, fee)

1) Retrieve sid's enrolled some course taught by Korth.

Enroll (sid cid ...) = Course (cid inst) — Korth

$\exists T_1 \{ s_1\ c_1 \quad \exists T_2 \{ c_1\ \text{Korth}$

$\{ T.sid \mid \exists T_1 \in Enroll\ \exists T_2 \in Course$
$(T_1.cid = T_2.cid \wedge$
$T_2.inst = Korth \wedge T.sid = T_1.sid) \}$

↑ There must be a free variable in result.

2) Retrieve sid's enrolled some course taught by Korth (and) some course taught by Navathe.

In R.A —

$\underbrace{\pi_{sid}(E \bowtie \sigma_{inst=Korth}(C))}_{P} \cap \underbrace{\pi_{sid}(E \bowtie \sigma_{inst=Navathe}(C))}_{Q}$

Note —

$P \cap Q \equiv \{x \mid x \in P \wedge x \in Q\}$

$P \cup Q \equiv \{x \mid x \in P \vee x \in Q\}$

$P - Q \equiv \{x \mid x \in P \wedge x \notin Q\}$

$$\{ T.Sid \mid \exists T_1 \in Enroll \; \exists T_2 \in Course \; (T_1.Cid = T_2.Cid \wedge T_2.Inst = Korth \wedge T.Sid = T_1.Sid) \wedge \exists T_1 \in Enroll \; \exists T_2 \in Course \; (T_1.Cid = T_2.Cid \wedge T_2.Inst = Nawathe \wedge T.Sid = T_1.Sid) \}$$

Ques: what is the english statement of —

① $$\{ T.Sid \mid \exists T_1 \in Enroll \; (T.sid = T_1.sid) \wedge \neg \exists T_1 \in Enroll \; \exists T_2 \in Enroll \; (T_1.sid = T_2.sid \wedge T_1.Cid \neq T_2.Cid \wedge T.Sid = T_1.sid) \}$$

$\Rightarrow P \wedge \neg Q$

$\Rightarrow P - Q$

$= \begin{pmatrix} Sid's\ enroll \\ atleast\ one\ course \end{pmatrix} - \begin{pmatrix} sid's\ enroll \\ atleast\ 2\ courses \end{pmatrix}$

All − (sid of different courses)

$\therefore$ Sid's enrolled only one course Ans

② $$\{ T.Sid \mid \exists T_1 \in Stud \; (T.sid = T_1.sid) \wedge \neg \exists T_1 \in stud \; \exists T_2 \in stud \; (T_1.age < T_2.age \wedge T.sid = T_1.sid) \}$$

$\Rightarrow$ All students — sid of student whose age is less than some other student

$\Rightarrow$ Sid's whose age is max Ans

Ques:

1) Retrieve Sid's enrolled every course.

In R.A :

$$\pi_{Sid\,Cid}(Enroll) \,/\, \pi_{Cid}(Course)$$

Sid's of Enroll (T_1) such that
for every course of Course (C)
there exist some record in
Enroll (T_2) with $T_2 \cdot Cid = C \cdot Cid$ and
$T_2 \cdot Sid = T_1 \cdot Sid$

Enroll (Sid Cid)		Course (Cid ...)		Enroll (Sid Cid)	
T_1	S_1	$\forall C$	C_1	$\exists T_2$	S_1 C_1
			C_2		S_1 C_2
			C_3		S_1 C_3

TRC:

$$\therefore \{ T.Sid \,/\, \exists T_1 \in Enroll\,(T.Sid = T_1.Sid) \wedge$$

$$\forall C \in Course\ \exists T_2 \in Enroll\,(T_2.Cid = C.Cid \wedge T_2.Sid = T.Sid) \}$$

Ans

2) Retrieve Sid's enrolled every course taught by KORTH.

In R.A – $\pi_{Sid\,Cid}(Enroll) \,/\, \pi_{Cid}(\sigma_{Inst = KORTH}(Course))$

We cannot use ques ① result just by adding an additional cond. $C.Inst = KORTH$

↓

as it checks for all courses and all courses must be taught by KORTH SI

C_1	KORTH
C_2	"
C_3	"
C_4	Navathe

S_1	C_1
S_1	C_2
S_1	C_3
S_1	C_4

~~SI~~ S_1 is only i.e enrolled all KORTH course but $C_4 \neq$ KORTH NEVER TRUE

$$\forall x \left(C(x) \longrightarrow S(x) \right)$$

x is Korth course — x should be enrolled by $T_1 \cdot Sid$.

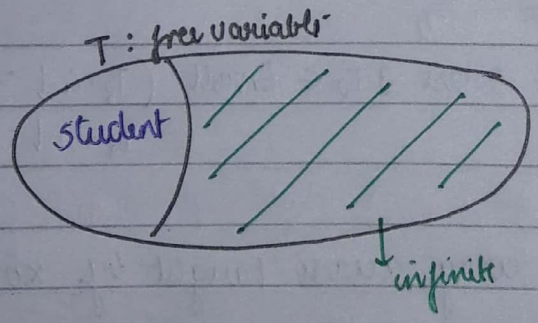
$$\{ T.Sid \mid \exists T_1 \in Enroll \, (T.Sid = T_1.Sid) \wedge \forall C \in Enroll \left(C.Inst = KORTH \longrightarrow (T_1.Cid = C.Cid \wedge T_2.Sid = T.Sid) \right) \}$$

Unsafe TRC Query —

TRC query with infinite set of records in result.

Eg - $\{ T \mid T \in Student \}$

↑ set of tuple (T) which does not belongs to student relation. (infinite records)



Expressive Power —

(Safe TRC Queries expressive power) $\approx$ (equal to) (Basic relational algebra queries expressive power)

→ **Basic R·A queries** — R·A queries which can be expressed using { π, σ, $\times$, $\cup$, $-$, ρ, η, $\bowtie_c$, $\bowtie$, $\ltimes$, $\rtimes$, $\bowtie$ }

Queries failed to be express using basic R·A :

- Sum of attribute val
  Avg of Attribute val
  Count of records / Count of attr. value

- Ordering of resulted records of query.