22/10/18

Recovery

Two properties -
ⓐ atomicity.
ⓑ Durability

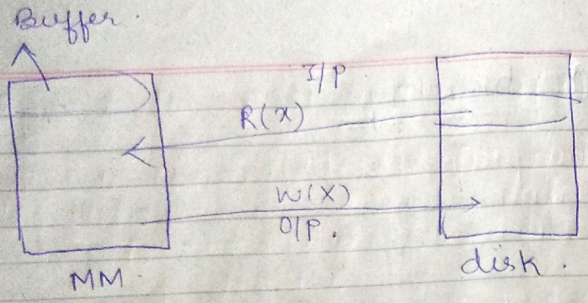
Types of failures

① Transaction failure
- logic (Eg:- divide by zero)
- system (Eg:- There can be deadlock)

② Disk failure
③ Raid (Redundant array of indepe-ndet disk).
↳ To recover from natural disaster

To recover data from natural disaster, we should store it on different geographic location.

④ Crash failure



Work area or work space is known as buffer.

W(X) → Disk (We immediately write it to the disk)
It is known as immediate modify

Output(X)
↳ commit / abort

If we write to the disk after commit or abort is known as deferred modification (delay)

If failure occurs somewhere b/w W(X) and output(X) → it is known as failure and here we need failure mechanism.

Log Records:

< T_i start >

< T_i update >

< T_i X_i V_1 V_2 > (changes X_i value from V_1 to V_2)

< T_i commit >

< T_i abort >

Undo → we do undo some T_i then it set the previous value / old value

Redo → It set to the new value.

Eg:-
< T_0 start >
< T_0 A 1000 950 >
< T_0 B 2000 2050 > ← crash occur somewhere here
< T_0 commit >
< T_1 start >
< T_1 C 700 600 >
< T_1 commit >

Before commit / redo, the value will be undo.
A = 1000 B = 2000

(II)

Suppose failure occurs after updation of C,

As T_0 is commited, so T_0 will be redo

so T_0 — Redo.

A = 950

B = 2050

As T_1 is not committed yet, so we will undo T_1

T_1 - Undo.

C — 700.

(III) If crash occurs after commit of T_1 so both T_0 and T_1 will be redo

A = 950 ; B = 2050 , C = 600

Checkpoints

Checkpoints will forcefully write the values to disk whether transaction is commited or not and we will add a checkpoint record <checkpoint>

Now we will chuck the active transaction if crash occurs (after the checkpoint)

~~R(A) ---------- B)~~

Eg: T_0 - -- T_{66} | T_{67}

T_{68} | T_{69} - T_{1000}

In T_{66} committed before checkpoint

T_{68} " " "

So T_{67} and T_{69} - T_{1000} (are in active state)

Series of steps

1. Suspend all the active transaction.
2. Add checkpoint record and maintain a log file.
3. Forcefully write to disk.
4. Resume all the transactions.

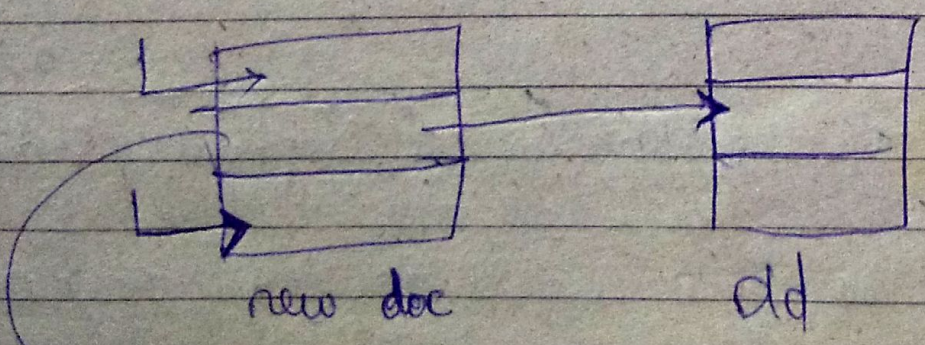
Note Book
Date
Page

Shadow Paging

We maintain the copy of the whole database. We will make changes to the new database, and we will see if all is written to disk. Then we will make the new database the current database and delete the old database.

We can't use this for very large database. So we use paging.

We will divide blocks into pages and maint-ain a page table. We will access blocks with page number and the page in which we make modification will be only copied and the one in which there is no modification will point to the old one.



The one with no modification points to the old one.

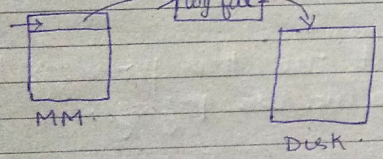
This is known as shadow paging.

But practically, this is also not good for large database.

23/10/18

Algorithms to recover data from log file

① Deferred update (delay update) :-

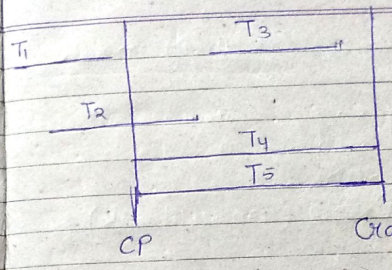


Just Before the commit statement is known as commit point.

As we reach commit point, then commit statement goes to log file and then all the updates are written to disk.

This algorithm is also known as No Undo/Redo.

Active state of transaction means no abort or no commit.



→ As T_1 is commited before checkpoint, we will simply ignore it.

→ T_2 and T_3 will be REDO as it lies b/w checkpoint and crash. It is commited so it will redo, set to new value.

Undo → set to old value.

Rollback → Undo + Transaction will restart with new time stamp.

→ T_4 and T_5 will be roll-backed as it is not committed.

Whenever crash occurs, we check from last statement before crash to the first checkpoint occurred.

Immediate Update.

Immediate

(UNDO/NO REDO)

(As soon as T is updated we will write it to the disk).

(As all the transaction

Commit

Here we will first write the log file to the disk and then we will write commit statement to the log file.

(UNDO/REDO).

→ In Immediate updation cost is very much

→ In the previous example,
T_1 will be simply ignored.
T_2 and T_3 will be REDO
T_4 and T_5 will be UNDO (no rollback).

→ We can't store/maintain the ~~data~~ log file in MM as it is volatile, when crash will occur log file will be deleted.
so we store log file in disk.
(Qus: But if crash occurs in disk then where the log file goes?)

→ If crash occurs while recovering mechanism (i.e, after one crash) then what happens?
(We understand that log file also maintains UNDO AND REDO operation)

→ Why do we need DBMS control manager to maintain DB files? Why can't OS deal with DB files also.

In OS if want to access a file, we can access a complete file (not a part of it).

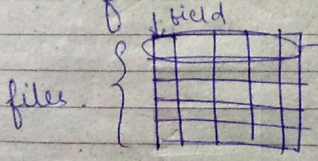
But in DBMS, we can access one record, multiple records (rows), one field, etc i.e, we can access a part of file. So OS can't deal with it.

DB
|
Files → OS can manage till here only.
|
Records
|
Field.

26/10/18

Indexing and File Organization

Database → collection of files
files → Records + fields.

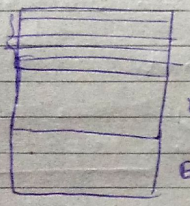


Average no:- of records in one block → Blocking factor (Bfr).

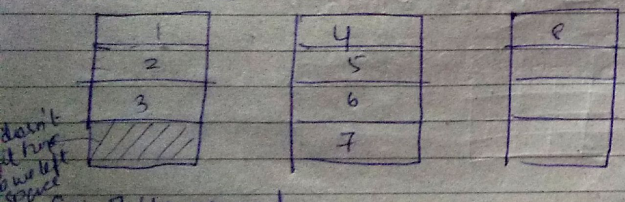
Note Book
Date
Page

$Bfr = \frac{\text{Block size}}{\text{Record size}}$ Imp

We divide database into blocks and each block consist of records. (We will not talk now in terms of files). (The block of database is mapped to block in a disk).



We can store records in block in 2 ways
① Unspanned.



Eg - of Unspanned.

Unspanned :- If one record doesn't fit in one block, we will waste memory and move it to another block.

But advantage is :- For one record, we have to access one block only.
This is used for fixed size records.

② Spanned

If our record doesn't fit, then we will store that much part of record which fit in that block and will move the rest to another block.
Advantage → No Memory wastage
Disadvantage → Have to access two block.

This is used for ~~one~~ records having no fix size.

Eg :-

URL	content

The content corresponding to each url is variable, so we will use spanned.

→ When we have ordered database (in ascending or descending order (eg eid)), so insertion cost is high, deletion is very low.
But when we delete, some memory space remains vacant, then database reorganize it

For searching of blocks, we will use binary search → Complexity ($\log b$) where b is no of blocks.

Unordered

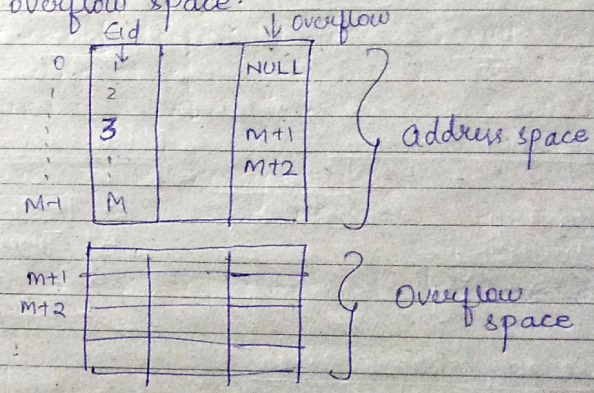
→ Not ordered in any order.
→ Now suppose our database is ordered by eid and we are using ename, then this is also un-ordered.
→ Data structure used in heap.
→ Linear search is used
→ Insertion is easy as it is unordered.
→ Here also we do reorganization periodically when we delete record.

Hashing

→ we have to design hash function in such a manner, if we are applying it on any hash field value then we get the exact index of that field value.
→ The o/p of the hash function is the index value.
→ Hashing → constant time complexity as it gives exact memory location.

Ordered and Unordered → can be spanned or unspanned (?)

→ Disadvantage → when at same space two records exist i.e., collision, then we do chaining.
For this we maintain a overflow pointer and overflow space.



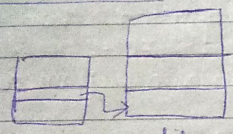
Suppose at Eid 3, we have two recordes, then we will write the index where the second record is stored in the overflow space, like here it is M+1.

→ External hashing → Apply hashing on Blocks. (Searching blocks).
→ Internal hashing → Apply hashing on records (Searching records in a block).

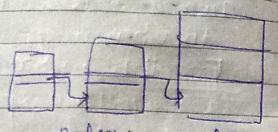
30/10/18

Note Book
Date
Page

Single level Index



Multi level Index



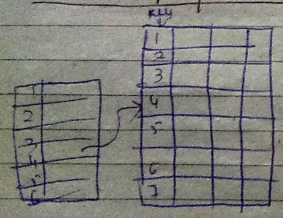
Single level Indexing

Primary
Clustering
Secondary

Multi-level

B Trees
B+ Trees.

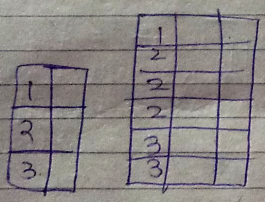
Dense and Sparse Index



Index file

Dense (every key value) ~~record~~ key

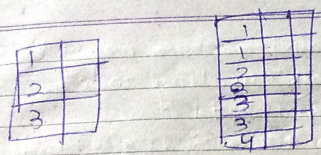
{ If for every value, there is an index then it is dense }



Sparse.

(As for every record there is no indexing)

Note Book
Date
Page



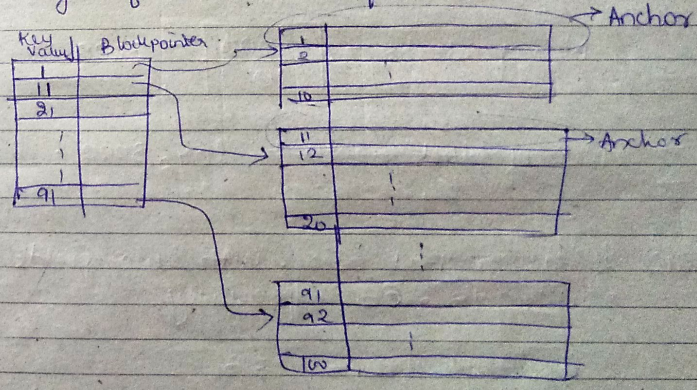
This is both dense and sparse. as for every key value there is an index so it is dense and for every record, there is no index so it is sparse.

Primary Indexing ~~Sparse~~

Adv

Prop of Index → Ordered (and ordered on primary key)

→ length of the record is fixed.



→ Anchor → The first record of every block.

→ Index are always ordered.

→ This is Sparse.

If we are directly accessing block → no of block access = $\log_2 b$; but by creating index, no of block access = $\log_2 b_i + 1$

Q There is an ordered file with 30,000 records, stored on a disk with block size 1024 bytes. files are of fixed size and are unspanned with record size = 100 bytes. Find the number of block access.

$$bf = \left\lfloor \frac{1024}{100} \right\rfloor_{floor} = 10.$$

$$\text{No of blocks} = \left\lceil \frac{30000}{10} \right\rceil_{ceil} = 3000.$$

$$\text{No of block access} = \log_2 3000.$$

(ii) If given that index, with two fields each of 9B, 6B respectively

Record size = 15B

$$bf = \left\lfloor \frac{1024}{15} \right\rfloor_{1024} = 68 \text{ records/index}$$

So no of indexes in index table

$$= \left\lceil \frac{30,000}{68} \right\rceil$$

$$= 45$$

No of block access = $\log_2 45 + 1$

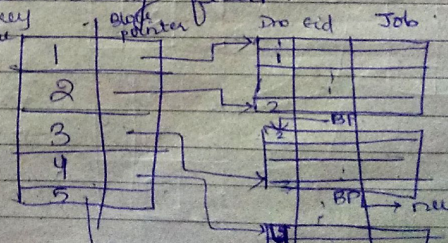
Disadvantage of primary

→ It can only be applied on ordered key field and not applicable to unordered field

Clustering Index

Ordered + nonkey

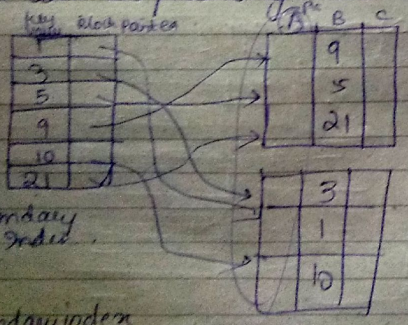
→ We will apply the search on ordered non-key field which is known as clustering field.



→ In index, we will do entry of every unique record.
→ It is both dense and spanned.
→ Here there is a BP (Block pointer) in the file pointing to the next block if the non-key value is same, otherwise NULL.
→ The index will access the first occurrence of non-key value in the block. The index 1 will point to the first occurrence of 1 in file.
→ No. of file access $\geq \log_2 i + 1$.

Secondary Index:

→ It is unordered → candidate key (Unique values but not ordered)
→ It can be on key or non-key.
→ Ordered on primary key.
→ There will be primary index on A here in this case.



Secondary Index

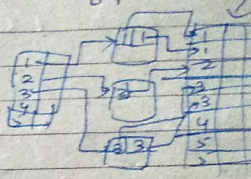
This is the case when we don't have duplicate B value (non-key).

No. of secondary index records = No. of B value

→ If we have duplicates, we will create a block for every unique value and to refer to this block we will make another block with ordered & unique value.

→ Now, for duplicates there will be use of primary indexing (doubt).

Q No. of records – 30,000
Record size – 100 Byte
Block size – 1024 Byte



$Bf = \left\lfloor \frac{1024}{100} \right\rfloor = 10$

No. of blocks $= \frac{30000}{10} = 3000$

As it is unordered, so we will apply linear search.
So avg no. of block access $= \frac{3000}{2}$
$= 1500$

Now,
If we created a secondary blocks with 9B, 6B
So one record size = 15B.

$Bf = \left\lfloor \frac{1024}{15} \right\rfloor = 68$

No. of records in index file = 30,000
(or entries)

No. of blocks in secondary index file
$= \frac{30000}{68} = 442$.

4+2+22+10

Note Book
Date
Page

* Index is always ordered, so we can apply binary search so no:- of searches = $\log_2 442 + 1$.

31/10/18

record { int eno 2; char name[22] 22; float sal 4; char dep[10] 10 } = 38 = Record size

No. of Record = 5000.
Block size = 512 bytes.

① No:- of block access without indexing.

② Bfr = $\left\lfloor \frac{512}{38} \right\rfloor$ = 13 records.

No:- of blocks = $\left\lceil \frac{5000}{13} \right\rceil$ = 385 blocks.

For ordered, binary search,
No:- of block access = $\log(385)$.

② Primary Indexing (15 bytes each)

$\frac{512}{15}$ = 34 record/index.

Note Book
Date
Page

No of index blocks = $\frac{5000}{34}$ $\left\lceil \frac{385}{34} \right\rceil$

= 12.

No:- of searches = $\log_2 12 + 1$

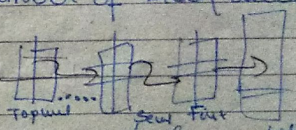
— x —

31/10/18

Multi-level Indexing

→ We do levelling till our all index comes in one block.
→ In that indexing in which all the index comes is known as top level index.
→ We do this to reduce the number of block access.

Q No of rec 30,000.
block size - 1024 byte
record size - 100 byte
Secondary index record size - 15 bytes.



bfr = $\left\lfloor \frac{1024}{15} \right\rfloor$ = 68. (Because here it is already given that secondary index is 15 bytes so we need not divide 100 bytes)

no:- of entries = 30,000.
no:- of blocks, $\left\lceil \frac{30,000}{68} \right\rceil$

4+8+22+10

Note Book

Date
Page

* Index is always ordered, so we can apply binary search
so no: of searches = $\log_2 448 + 1$.

31/10/18

record { int eno 2; char name[22] 22; float sal 4; char dep[10] 10 } = 38 = Record size

No: of Record = 5000.
Block size = 512 bytes.

① No:- of block access without indexing.

② $Bfr = \left\lfloor \frac{512}{38} \right\rfloor = 13$ records.

No:- of blocks = $\left\lceil \frac{5000}{13} \right\rceil = 385$ blocks.

For ordered, binary search,
No:- of block access = $\log(385)$.

② Primary Indexing (15 bytes each)

$\frac{512}{15} = 34$ record/index

Note Book

Date
Page

No:- of index = $\frac{5000}{34} \left\lceil \frac{385}{34} \right\rceil$

= 12.

No:- of searches = $\log_2 12 + 1$

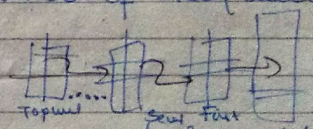
— x —

31/11/18

Multi-level Indexing

→ We do levelling till our all index comes in one block.
→ In that indexing in which all the index comes is known as top level index.
→ We do this to reduce the number of block access.

* No of rec 30,000
block size - 1024 byte
record size - 100 byte
Secondary index record size - 15 bytes



$bfr = \left\lfloor \frac{1024}{15} \right\rfloor = 68$. (Because here it is already given that secondary index is there so we need not

no:- of entries = 30,000.
no:- of blocks, $\lceil 30,000 \rceil$

= 442 :

As, 442 is also more blocks.
so, we will do ~~secondary~~ 2nd level indexing

So no:- no:- of blocks = $\left\lceil \frac{442}{68} \right\rceil = 7$.

Now 7 is also more than 1.
So we will go for 3rd level indexing.

So no:- of blocks = $\left\lceil \frac{7}{68} \right\rceil = 1$.

So now here, 3rd level index is our top level index. as ~~it~~ here there is only 1 block.

[In the original file; no:- of records per block = ~~200~~ 1024/10 = 20

no:- of blocks = $\frac{30000}{10} = 3000$ blocks.

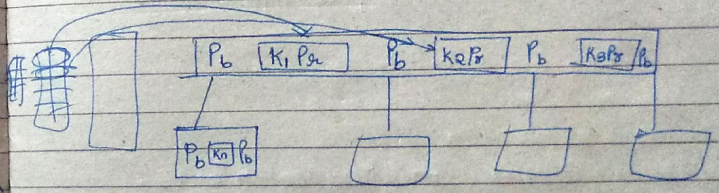
So from 3000 blocks (original file) → 442 blocks (1st level) → 7 blocks (2nd level) → 1 block (top level)

No:- of block access = 4. (1+1+1+1)

No:- of block access = $t + 1$ → file access
(top level)

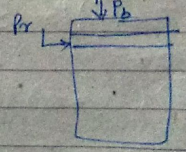
→ Here we will have problem for insertion & deletion. So to overcome this we move towards dynamic multi-level indexing which is implemented through B and B^+ tree.

B-tree → structure of internal node & root node is same.



Pb → Pointer pointing to a particular block
Pr → Pointer pointing to a particular record

No:- of blocks < No:- of records.
Size of block pointer < Size of record pointer



Note Book
Date
Page

Order of the B tree = maximum no. of children
Minimum no. of ~~tree~~ children = 2
of root
max = P...

root
For root, key values can be :-
min = 1
max = P-1

node
No. of children, min = P/2 max = P.
No. of key values, min = ⌈P/2 - 1⌉ max = P-1

Q Consider a B tree with key size 10 bytes, Block size = 512 bytes, data pointer (P_r) size = 8 bytes and Block pointer size = 5 byte. Order of B tree

~~No. of block pointers =~~

$nP_b + (n-1)k + (n-1)P_r \leq$ Block size

$\Rightarrow n \times 5 + (n-1)(10+8) \leq 512$

$\Rightarrow 5n + 18n - 18 \leq 512$

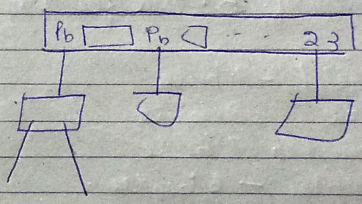
$\Rightarrow 23n \leq 530 \Rightarrow n \leq 530/23$

$\Rightarrow n \leq 23$...

$n = 23$

02/11/18

Q Order of B tree = 23
Then how many maximum index records can be stored in 4 levels of B tree.



Total index records in 4 levels =

$22[1 + 23 + 23 \times 23 + 23^3]$

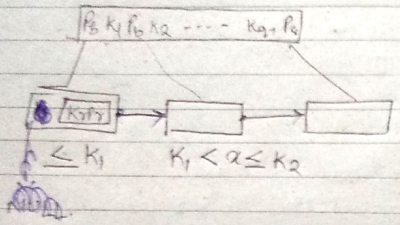
$= 22[1 + 23^1 + 23^2 + 23^3]$

$x \rightarrow (y)$ $y \subseteq x$
$y \subset x$

Note Book
Date
Page

Note Book
Date
Page

B+ trees



Q Find the order of B+ tree for the given data.
Search key field - 9 byte
Block size 512 byte
Record pointer 7 byte
Block pointer - 6 bytes.

Internal node

$n \times 6 + (n-1) \times 9 \leq 512$

$\Rightarrow 15n - 9 \leq 512$

$\Rightarrow 15n \leq 521$

$\Rightarrow n \leq \frac{521}{15}$

$\Rightarrow n \leq 34.$ (approx)

$n = 34$ Order for internal node

Study views from lab

Note Book
Date
Page

Leaf node

$(n-1)(k+P_r) + n \times P_b \leq BS$

$(n-1)[7+9] + n \times P_b$

$\Rightarrow (n-1) \times 16 + n \times 6 \leq 512$

$\Rightarrow 22n - 16 \leq 512$

$\Rightarrow 22n \leq 528$

$n \leq \frac{528}{22}$

≤ 24

$n = 24$ Order for leaf node
→ pointing to next node

$\Rightarrow m(K + P_r) + P_b \leq BS$

$\Rightarrow m(7+9) + 6 \leq 512$

$\rightarrow 16m + 6 \leq 512$

$\rightarrow 16m \leq 506$

$\rightarrow m \leq \frac{506}{16}$

$\leq 31.$ (approx)

$m = 31$ one order for leaf node