Indexing

Agenda

Why & what of indexes?

How indexes work

Indexes on multiple columns

Cons of Indexes

Indexes on Strings.

select * from film where film_id = 100;

↓

TABLE SCAN : row by row ⟶ O(N)

B1 B2 B3 B4

↑

202

HM : film_id, address .

↓ O(1) .

Index , indexing

select * from users where name = "MOHIT"; }

↑ ↑

create an index [Key : name , value : row.]

We can have duplicate name

HM < String, int > ✗

HM < String, List >

Mohit	3, 7, 13, 18
Akash	1, 10, 16, 18

Select * from film where release-year between (2016, 2023);

[Key: release-year, value: List< >]

2019
2023

range

2016
2017
2018
2019
2020
⋮

HM's are helpful → ordered → $\log_2 N$

HM + ordered

↑

TreeMap / ordered-map

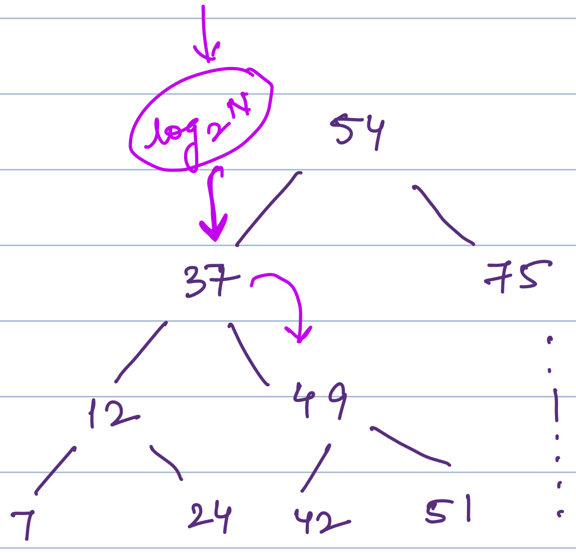
↓

BBST: Balanced Binary Search Tree.

↓

O(H)

O($\log_2 N$)

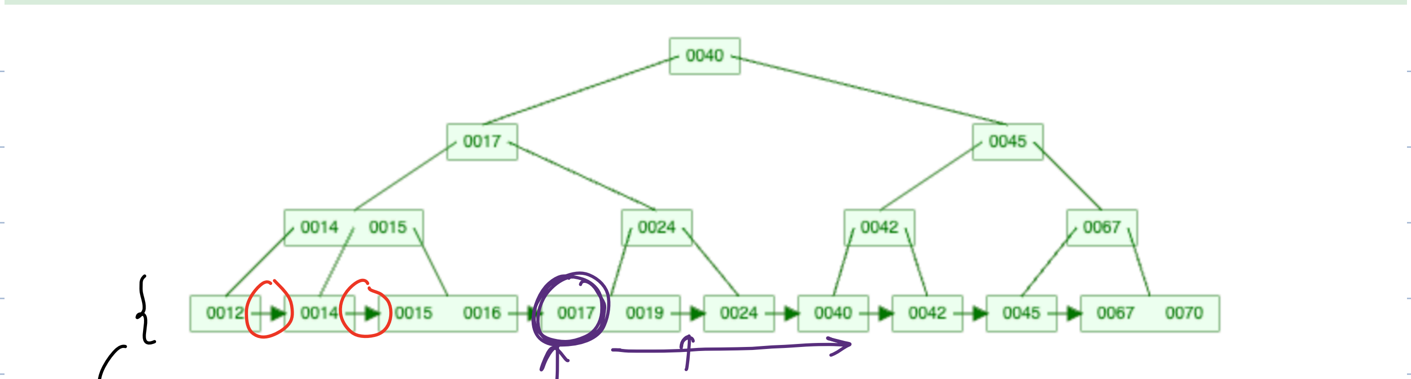


inorder

B / B+ Trees.

↓

decrease height

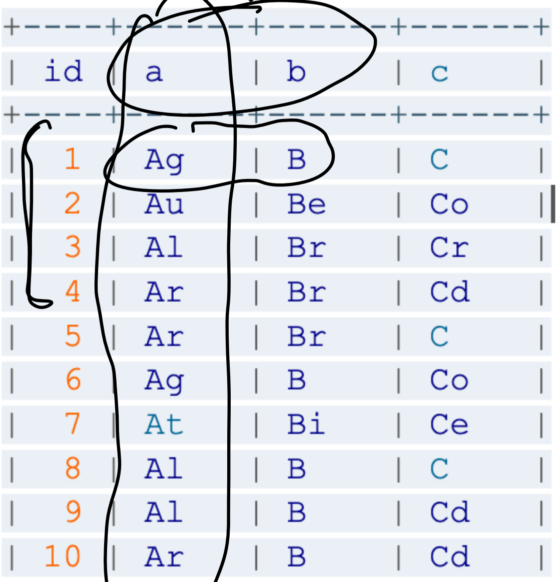


① all elements

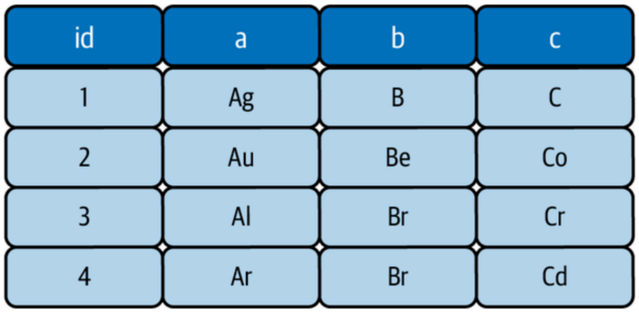
② leaf node maintains a pointer to next leaf Node.

mysql : tables : indexes : clustered indexes

indexes on primary keys



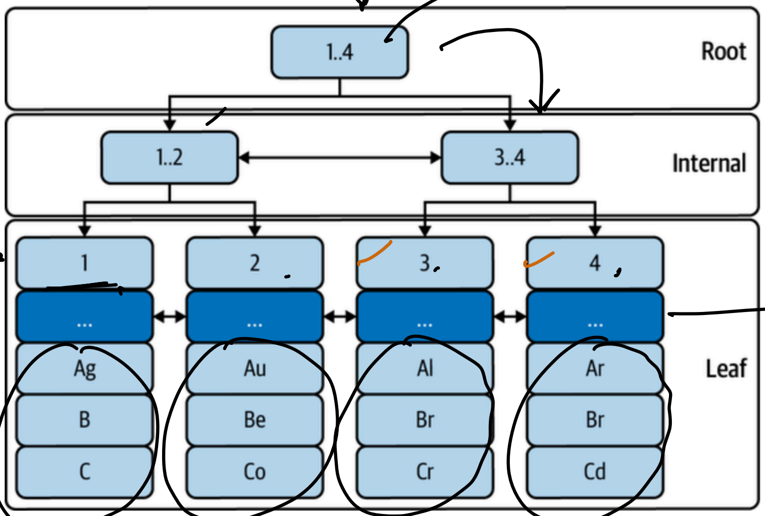
```
+----+----+-------+-------+
| id | a  | b     | c     |
+----+----+-------+-------+
|  1 | Ag | B     | C     |
|  2 | Au | Be    | Co    |
|  3 | Al | Br    | Cr    |
|  4 | Ar | Br    | Cd    |
|  5 | Ar | Br    | C     |
|  6 | Ag | B     | Co    |
|  7 | At | Bi    | Ce    |
|  8 | Al | B     | C     |
|  9 | Al | B     | Cd    |
| 10 | Ar | B     | Cd    |
```



id	a	b	c
1	Ag	B	C
2	Au	Be	Co
3	Al	Br	Cr
4	Ar	Br	Cd

Ag Au Al Ar

primary key



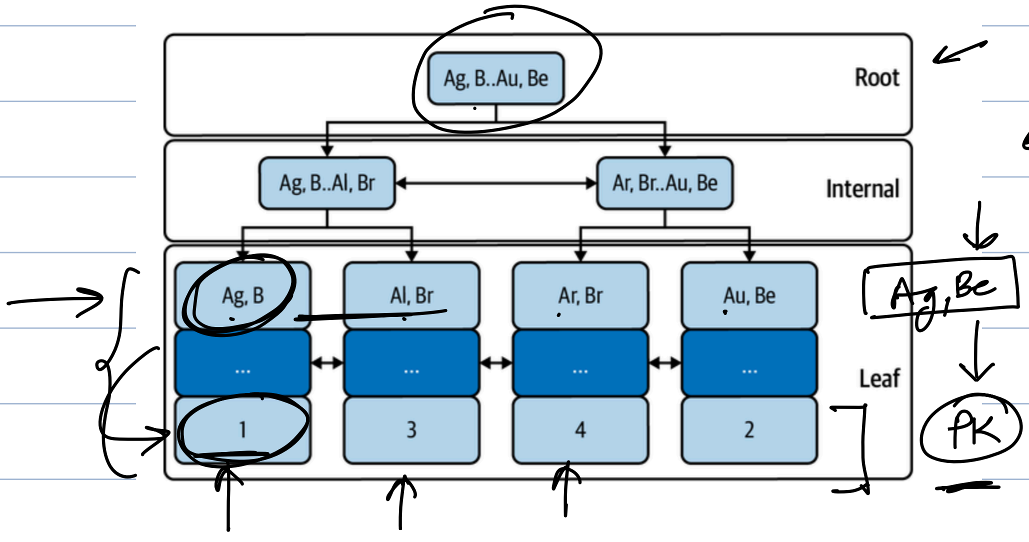
metadata.

select * from film
order by film_id

We can create our own
indexes : Secondary , Non-clustered index .

select * from symbol where a = 'Ag' and b = 'Be';

column → PK .



Ag, Al, Ar, Au

Multiple columns

where film_id = __ ;

order by film_id ;

customers

email, PAN

when email = "____"

& PAN = ____ ;

↓ ↑

complete table scan.

indexes on PAN → unique

brand ———— brand, category.

category

brand, category. X

rows = 10,000

brand = 10 , categories = 100

$\frac{1000\cancel{0}}{1\cancel{0}}$ $\boxed{1000}$

$\frac{100\cancel{00}}{1\cancel{00}}$

100 produ.

(A, B)] diff indexes

(B, A)

↓

order matters.

{ marks, id

id, marks }

idx → (f_name, last_name)

idx → (actor_id) ✗

PRIMARY

complete data

select * from actors
where f_name = ' '
& l_name = '_';

first_name

select ^ from actors
where f_name = ' '
& l_name = '_'

covering indexes.

I actor_id

II actor_id, email

III last_name, email

LEFT PREFIX

actor_id → I

last_name → III

last_name, actor_id → I

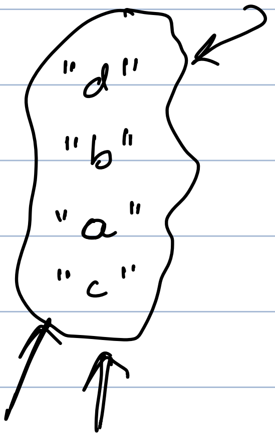
email → None of the above.

actor_id, email

3	"a"
1	"d"
2	"b"
4	"c"

→

1
2
3
4



a, b, c, d, e

I a

II a, c

III a, c, b

IV a, b, c

c and b X

c X

a and b

first_N, L_Name → first_name

X first_Name →

→ a use-less index

last_name

↓

a new idx will be required -

where a = ____ and b = ____ ;

a = ____ OR b = ____

↑ ↓

a

b → x True

~~(a, b)~~

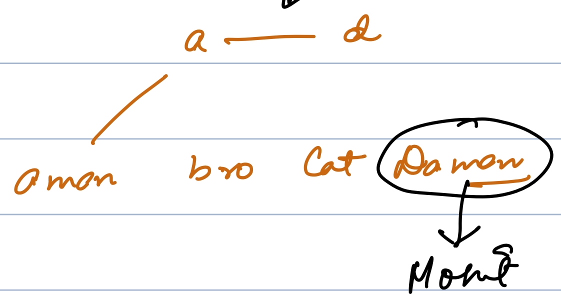
(a, c) → a or b

CONS OF INDEXING

① indexes occupy space

② C R U D → writes can become slower

U ↓ update



12 columns
↓
12 indexes ✓

insert

[BIT MAP INDEX]

low cardinality :- no of unique values.

↓

selectivity : $\frac{\text{no of unique}}{\text{total}}$ → 0-1

drink
- cofee
- tea

→ 1 M row

10^6 rows.

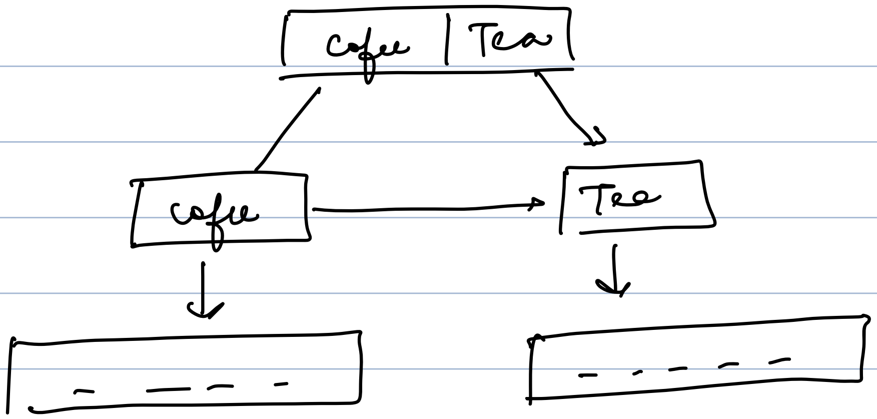
2

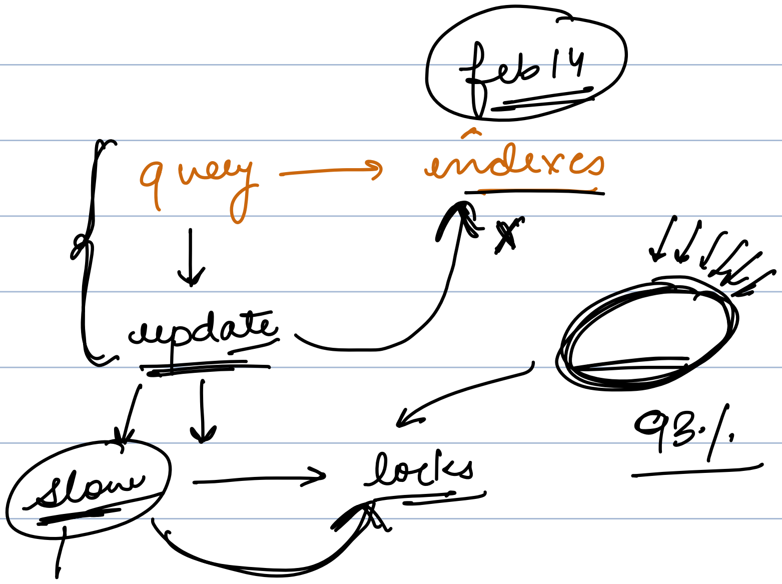
$\frac{2}{1\,M}$

→ 0.5 M → O(n/2)

O(n) ✗

select * from bevey where drink = "tea";


cofee | Tea
cofee
Tea


feb 14
query
indexes
update
slow
locks
93%

INDEXING ON STRINGS

① bad in comparison

② space ⇑

10

______ @gmail.com }

where email = "______";

⇑

index ✓

10 M records

↳ 9M

□ @gmail.com

create index idx_email on customer(email(7));

2B records

__ → 26

__ , __ → 26 * 26 = 676

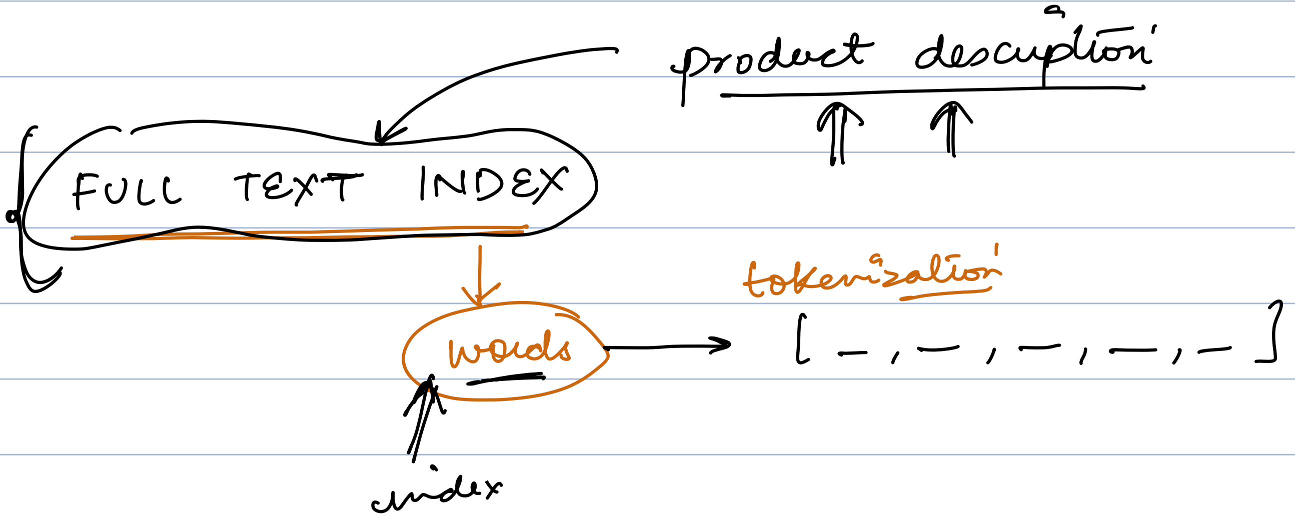
__ __ __ → 17576

$26^4 = 4*10^5$

$26^5 = 10^7$

$26^6 = 3*10^8$

$26^7 = 8*10^9$



product description

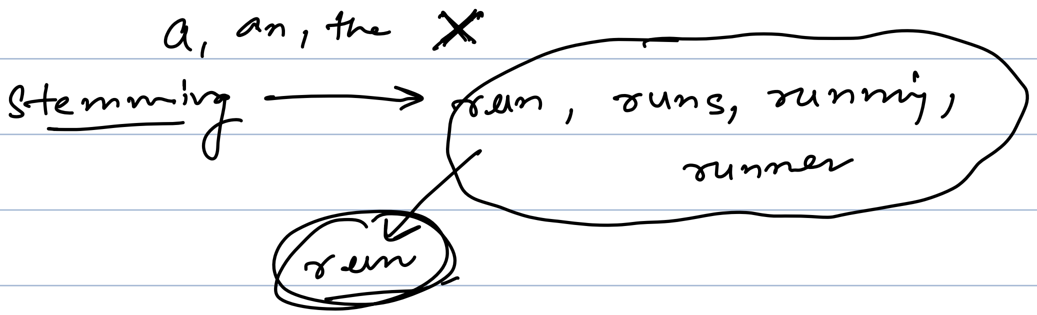
FULL TEXT INDEX

words

tokenization

[_ , _ , _ , _ , _]

index



a, an, the ✗

Stemming → run, runs, running, runner

run

MONITOR PERFORMANCE