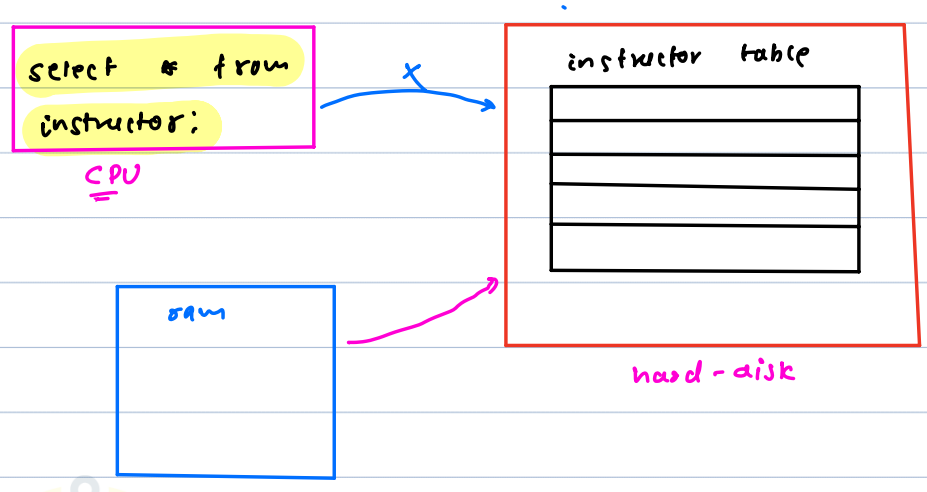
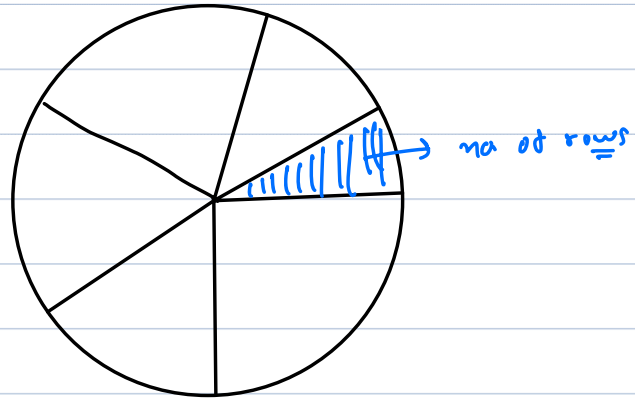
indexing



- cpu can't directly interact with hard disk because of speed difference.
- cpu interact with ram and ram get the information from harddisk (disk I/O)



each data block contains some number of rows.

- any table is sorted by default on primary key.

	i_id	name	salary	d_id	
a1	101	A	10k	205	data block 1
a2	102	B	20k	203	
a3	103	C	15k	201	
a4	104	D	35k	203	
a5	105	E	50k	201	data block 2
a6	106	F	40k	203	
a7	107	G	45k	204	data block 3

AlgoPrep

get all the instructor information where d_id is 203

- if we have 10,00,000 rows ?

the number of disk I/O (fetch the data from the disk) = no. of rows

10,00,000

(get the instructor whose i_id is 104 :) multiple times

(10,000)

no. of query = 10,000

i_id	add
101	a1
102	a2
103	a3
104	a4
105	a5
106	a6
107	a7

O(1)

	i_id	name	salary	d_id	
a1	101	A	10K	205	data block 1
a2	102	B	20K	203	
a3	103	C	15K	201	
a4	104	D	35K	203	
a5	105	E	50K	201	data block 2
a6	106	F	40K	203	
a7	107	G	45K	204	data block 3

AlgoPrep

203

i_id	add
201	a3
201	a5
203	a2
203	a4
203	a6
204	a7
205	a1

index table

we create this table in ram.

O(1)

total we access the disk 3 times.

index

* its provide a quick path to locate any specific row or set of rows, without scanning entire table.
- its reduces the number of disk I/O required to fetch data and lead to faster query performance.
- when the queries involves filtering, sorting, order by, group by, or joining column.
- it can be create on multiple tables as well.
- especially for large datasets.

- indexes required additional storage spaces.
- required regular updating and maintanance to remain accurate for write operations (create, update, delete)

AlgoPrep

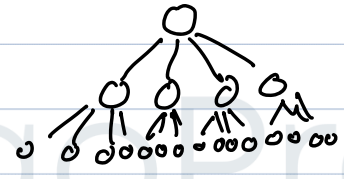
- we make an extra copy of index table in disk.

- its stored values in sorted on basis of index column.
- it gives faster retrieval of data.

treemap, Hashmap

B / B+ tree (balanced tree) (complexity < log(n))

neigh of the tree



AlgoPrep

create index idx_instructor_d-id
on instructors(department_id):

drop index idx_instructor_d-id
on instructors.

break till 9:15

lets suppose index column data-type is string

select * from
instructors where inst_name = "deepanshu"

if we create index table using only single character.

[a, b, c ..... z] 26 1,00,000

[a → aman
a → akshay
b →
b →
.
c →
.
.
z]

(d → (_, _, _, _) ;)
deep, divyam, deekshu ...

50,000

create index idx_student_name
on students (name (1));

* it is used for uniform distribution of data on starting character.
* it has a smaller index size compare to other.

* if data is not uniform distributed then in that case it takes more time.
* we will interact with disk 50000 time

select * from
instructors where inst_name = "deepanshu"

[26 × 26]

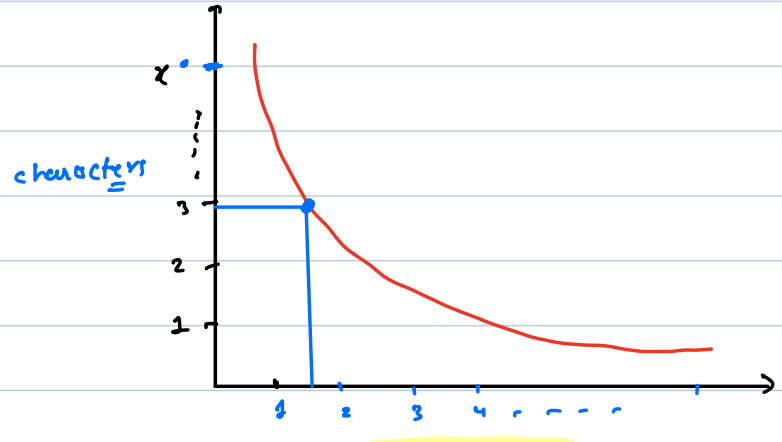
(2) if we create index using two characters.

aa
ab
ab
⋮
da
⋮
de
df
⋮

de [ ] → 5000

* it is used for uniform distribution of data on starting character.
* it has a larger index size compare to first character indexes.

* we will interact with disk 5000 time



full text index

camera

Q) we create a table of Products which have P_id, P_name, description

P_id	P_name	decription
	(2-3)	(50-100)
	(2-3)	(50-100)
	(2-3)	(50-100)

```
create fulltext index idx_Product_name_desc
on Product (name, description);
```

find all the products where either name or description contains camera or digital

```
select *
from Product
where match (name, description)
against ('camera digital' in boolean mode);
```

- + stands for And (camera +digital)
- − stands for Not (camera +digital −security)
- (no operator) implies OR

AlgoPrep

```sql
show indexes in students;
show indexes in products;
explain select *
from students
where st_id = 50;

explain select *
from students
where marks = 50;

create index idx_students_marks
on students(marks);

drop index idx_students_marks
on students;

create fulltext index idx_product_name_desc
on products(name , description);

select * from products
where match (name , description)
against ('photography -lens' in boolean mode);

select * from products
where match (name , description)
against ('+photography +camera' in boolean mode);
```