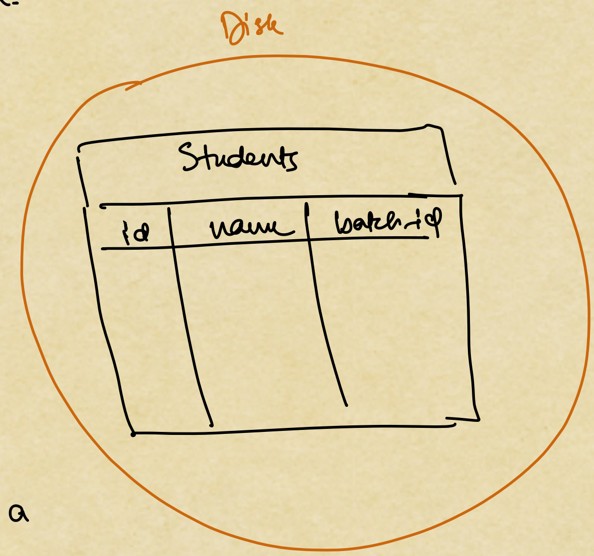
⇒ Indexing

DB stores data on a disk.



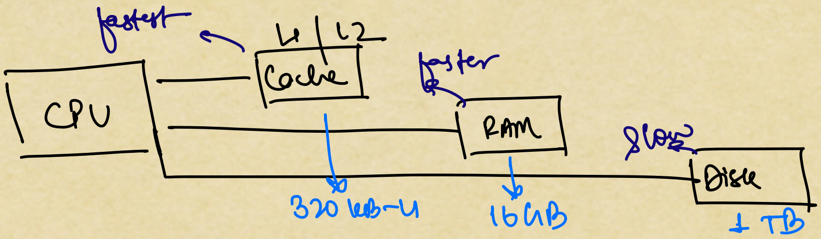
```
Select * from Students
where batch_id = 2;
```

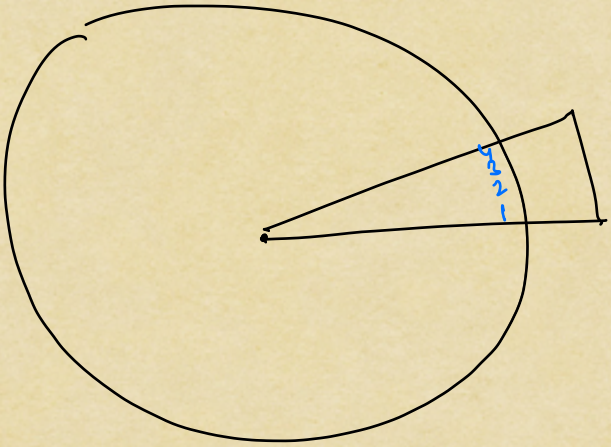
- OS can't directly operate on a disk
- Content from the disk is brought into memory and then appn works on that data.

⇒ If CPU were to directly work on the disk it will be wasting a lot of its time.

→ CPU makes an I/o call to disk for bringing the data to memory

→ while the data is being copied. It will do something else.





⇒ 1, 2, 3, 4

Query ⇒ 3

* A table is always sorted by its pk ⇒ default

Students → 1M rows

id	name	batch_id	psp	address
1	A	3	80	—
2	B	3	90	—
3	C	2	60	—
4	D	4	75	—
5	E	1	82	—

RAM & compare

Ram & compare

Ram & Comp

R&C

R&C

Query ⇒ select * from Students where batch_id = 3;

result ⇒ 20 Students

total disk access = 1M

useless " " = 1M - 20

pk

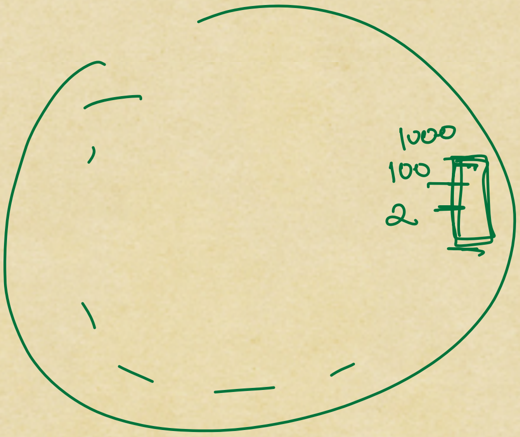
Query => select * from Students where student_id = 3;

Students → 1M rows

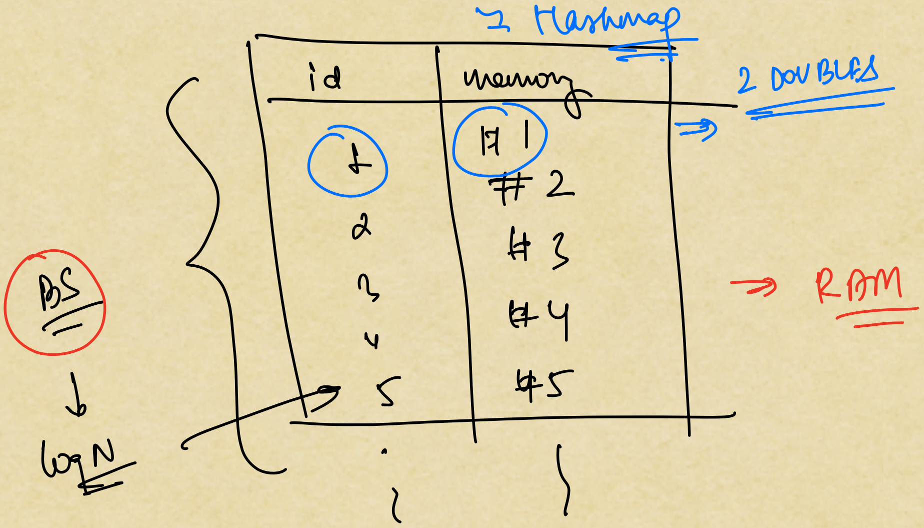
id	name	batch_id	psp	address	memory address
1	A	3	80	—	#1
2	B	3	90	—	#2
3	C	2	60	—	#3
4	D	4	75	—	#4
5	E	1	82	—	#5

* because the table is sorted by pk, once we hit the condition (we get matching data) we stop & return

* still worst case can be 1M



=> Create a table that stores id of a row and memory address of a row, sorted by id.



id	memory
1	#1
2	#2
3	#3
4	#4
5	#5

1M rows ⇒ Select * — — where id = 15;

↓ per

disk fetches

CASE 1 ⇒	No sorting	No address table	⇒	1M
CASE 2 ⇒	Sorting	No address table	⇒	15
CASE 3 ⇒	Sorting	address table	⇒	1

* We want to reduce the no. of disk fetches.

eg ⇒

Students

id	name	batchid	psp	phone No.
1	A	3	80	123 → #1
2	B	3	90	124 → #2
3	C	1	60	678 → #3
4	D	2	75	679 → #4
5	E	1	85	576 → #5

Select * from Student where phoneNo = K;

↓

table → sorted by phone No.

phoNo.	address
123 →	#1
124 →	#2
678 →	#3
679 →	#4
576 →	#5

⇒ query ⇒ Select * from Student where batch.id = 3;

batch id	address
3	#1
3	#2
1	#3
2	#4
1	#5

↓

sorted by batch-id

batch id	address
1	#3
1	#5
2	#4
3	#1
3	#2

ex => Select * from Students where psp is between 10 and 35;

psp	address	
0	#1 #6 #7	
10	10 9 8	→ ≥10 & <20
20	2 4 5	→ ≥20 & <30
30	31 63 32	→ ≥30 & <40
40	52 62 64 47	
50		
60		
70		
80		
90		
100		

⇒ **Indexes :**

→ Prevent unnecessary disk fetches.

→ leads to faster queries

→ Should we create indexes always?
↓
No. it depends.

⇒ C | U | D on table ⇒ indexes will also need to be updated.

⇒ if we do any write operations, index needs to update

⇒ indexing makes read faster & writes slower.

⇒ index tables are actually in disk, leads to increase in memory usage.

DS used for indexes ⇒ | B Tree | B+ Tree |

H/W ⇒ read about B Trees

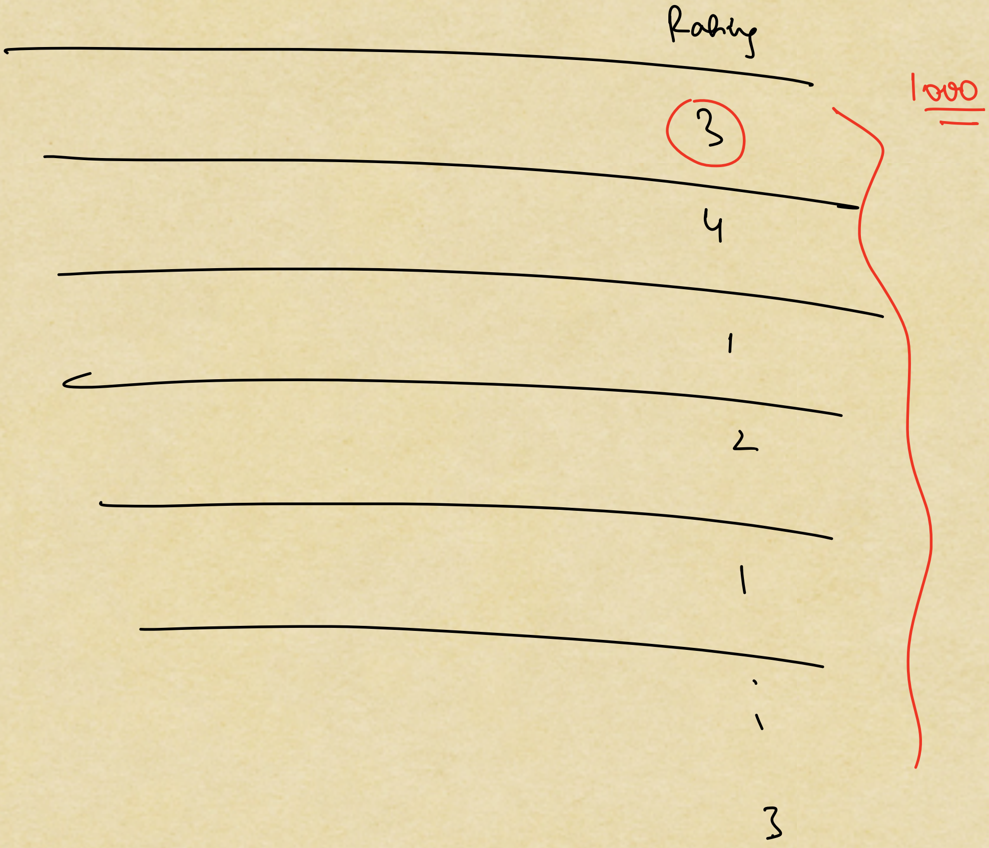
⇒ When to create indexes:-

i) Do not create indexes at the beginning

ii) Do performance testing

iii) Create indexes for queries that are used a lot and are slow

iv) Create indexes by access patterns and not predictions.



>200000

↓

1000

index

Rating	Addre
3	#123
4	
5	
|	
|	
|	