SQL / Relational Architecture

Server → Database

Web Tier — Db Tier (SQL)

SQL follows Master-Slave Architecture

User

userId (PK)	name	email
101	Aditya	xyz
102	Rohit	ert
103	Mukesh	wer

Horizontal Scaling --> Sharding

User --> millions

101 - 10001 shard1

Server — write → Master; Master → Slave1, Slave2, Slave3; Server — read → Slave3

Views. --> Virtual Tables

SQL DB

User

userId (PK)	name	email
101	Aditya	xyz
102	Rohit	ert
103	Mukesh	wer

Relational DB. --> Rows and Columns

Logical Arrangement

Stored not in Volatile memory like RAM

Stored in permanent memory (Disc)

Hard disk structure — Data Block, Sector

DBMS → Data Page

Table

userId (PK)	name	email
101	Aditya	xyz
102	Rohit	ert
103	Mukesh	wer

1 Million Rows

tables are stored in the form of DATA PAGES in the memory

tables are divided into data pages

Data Page: Headers → 96 bytes; Data Records (rows data are stored here) → 8060 bytes; offset → 36 bytes

divided in 96 + 8060 + 36 bytes = 8KB

8 KB = 8192 Bytes — generally one data page can store 8l

Headers and offsets are metadata and data records store the rows

row = 32 bytes

suppose each row is of 32 Bytes (depends on the no.of columns)

1 data page can hold 251 rows

1 Data Page = ~251 rows. (8060/32

Total page. = 3984 pages → for 1 million rows

total data pages required

Data Page

Headers — stores → Page Number, checksum (used for error detection)

Data Records

offset — stores → Array of indexed rows

Data pages are managed by DBMS.
But practically, All data pages will be stored in memory.

and in memory data blocks are present where actual data is store (check the hard disk structure) → means data block stores data pages

1 Data Block --> 8 KB - 64 KB

data blocks are mapped with data pages

say DBMS knows the address of the data blocks but do not manage the content in data blocks (managed by hard disk)

If 1 Data block = 8KB
1 Data block = 1 Data Page

Data Block :
It is the smallest possible value we can expect from an IO operation.

user

userId	name	email
101	Aditya	xyz
103	Rohit	ert
102	Mukesh	wer

→ Data Page

Select * from user
where userId = 101;

server ⇄ SQL DB ⇄ hard Disk

Data Block 1 - page 1, Page 2
Data Block 2 - page 3, page 4
Data Block 3

data pages → unclustered way

O(N) time complexity --> 10 million records

not good

Data Blocks are managed by Hard disk.
DBMS manages Data Pages.

DBMS has a mapping table (User) to understand which data page is stored in which data block

say DBMS knows the address of the data blocks but do not manage the content in data blocks (managed by hard disk)

Data page 1 ---> Data block 1
Data page 2 --> Data block 1
Data page 3 --> Data block2
....

this is how data is stored in memory without indexing (i.e. unindexed table)

Data page 3984 --> Data block 1992 → to much query to get the data at page 3984 hence high time complexity

Indexing :
used to increase the performance of our Db Queries.
--> Making Queries run faster.

O(N) --> Reduce my search result.

DSA --> Binary Search /Binary Search Tree (BST)
B+ Trees are used for DB indexing.

O(N) --> O(logN) → for B trees

O(N) -> O($\log_F$N) → For B+ trees

Indexing → Clustered Indexing, Non-Clustered Indexing

B Trees

9, 11, 7, 3, 23, 5, 16

root [P | 9 | P] (P - Pointer); 9 → 7, 11; 7 → 3, null; 3 → 5; 11 → 23; 23 → 16

```
BST
{
    int data;
    BST p1;
    BST p2;
}
```

M-way
M = 2
Keys = 1 (M-1)

BST --> Binary Seach Tree (2)
TST --> Ternary Search Tree (3)
M- way Trees --> M children (M) children

root [P | 3 | P | 5 | P] (k)

M-way tree
M = 3
k = M-1 (3-1) = 2 (no.of keys)

```
{
    int data1;
    int data2;
    Btree p1;
    Btree p2;
    btree p3;
}
```

B Trees

[4 | 5] Node

M-Way tree
M = 3
keys = M-1 = 2

[9 | 11] Root node → sorted order (m-order); children [3 | 7], [23 |]

B-Tree Split up

9, 11, 7, 3, 10, 23, 5, 16, (12, 34, 1, 89)

[9 | 11]

[9 | 11] → [3 | 7], [10 |], [23 |]

[10 | 19] → [5 | 9], [11 | 16]; [5 | 9] → [3 | 4] → [7 | 8] → [ ]

B+ Trees → leaf nodes connected; Intermediate nodes also store parent left value

[12 | 19] → [5 | 10], [12 | 13]; [5 | 10] → [3 | 4] → [5 | 7] → [ ]

B-Tree
B+ Tree
Code

Indexing → B+ Trees

user

userId (PK)	name	email
19	Aditya	xyz
25	Rohit	ert
30	Mukesh	wer

1 Million records

→ Data Page (data page): 101, Aditya, xyz; 103, Rohit, ert

B+ tress of userId → M_way (m ?)

[17 | 25] → [1 | 6], [17 | 19], [25 | 30] → Data page 1: 19, Aditya, xyz; 25, Rohit, ert; 30, mukesh, wer

250 values

this is leaf node (not intermediate note) this connections(beteen leaf nodes) are used to do binary search in B+ trees since the values in the nodes are sorted

select * from user
where userId = 19;

DBMS
data page --> Data block mapping

asymptotically B and B+ trees order is log(N), but B+ trees is better than B trees , because traversing in B+ trees is different than B trees
if we are deleting any node in B+ trees then the node is only deleted in leaf node and not in the intermediate node(but increases space complexity) unlike B trees which reduces the indexing speed in B trees

B+ Tree

Range Query --> 101 - 205

we have the range in the sorted order in B+ trees hence we can get the data quickly

but in B trees you are searching on every node for range query

Data Page: Headers; Records: 19, Aditya, xyz; 30, Rohit, ert; 25, mukesh, wer; offset: 19 | 25 | 30

id → user

userId	name	email
19	Aditya	xyz
30	Rohit	ert
19	Mukesh	wer

1 Million records

in data records data is inserted in same order as it in inserted in the table, may be sorted or unsorted depends on the insertion in the table
Hence offset in the data page keeps the userIds in a sorted manner and userIds in the in the offset are mapped with their corresponding data in the data records of the page

Indexing → Clustered, Non Clustered

Clustered: Leaf nodes of B+ Trees store complete row information

Non Clustered: leaf nodes of B+ trees store only id information (page no.)

i.e just gives the page number of the data page we need to search again for the data in records using offset

clustering can be done only to the ids whose primary keys are unique and not null

Indexing :
Clustered vs Non-Clustred ← offsets plays the role
--> There can be only 1 clustured index per Table.
--> There can be multiple non clustred index per table.

sorting table on the basis of any one column of the table and the values in the column should not be repeating which may create conflicts

hence clustered index per table

--> What is the issue to index each column??
--> It will require a lot of space.

space complexity is very high for indexing for storing tree data and all which consumes lot of space

hence database team has to give permission to the developer for indexing which column

NO-SQL DB
Mongo Db --> B+ Trees
Cassandra DB --> LSM Tree

for indexing