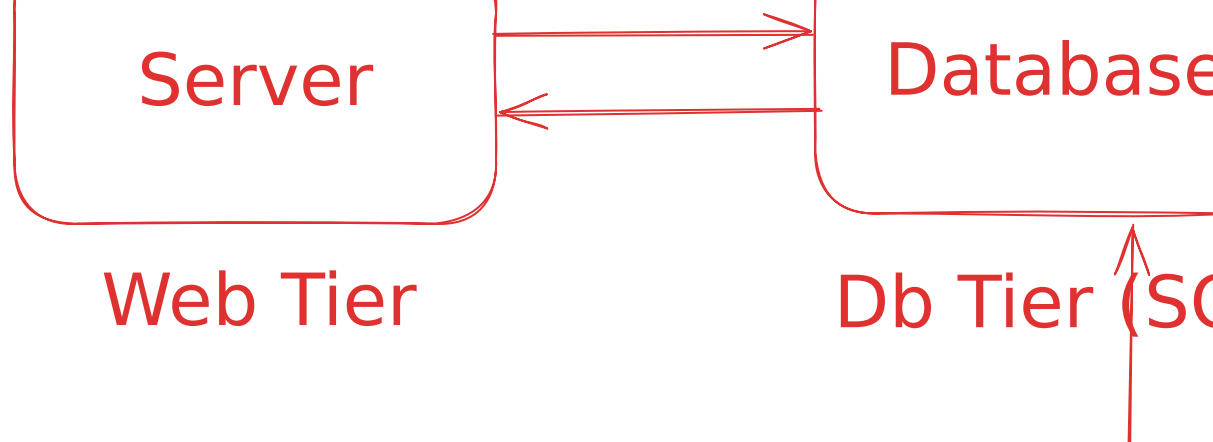
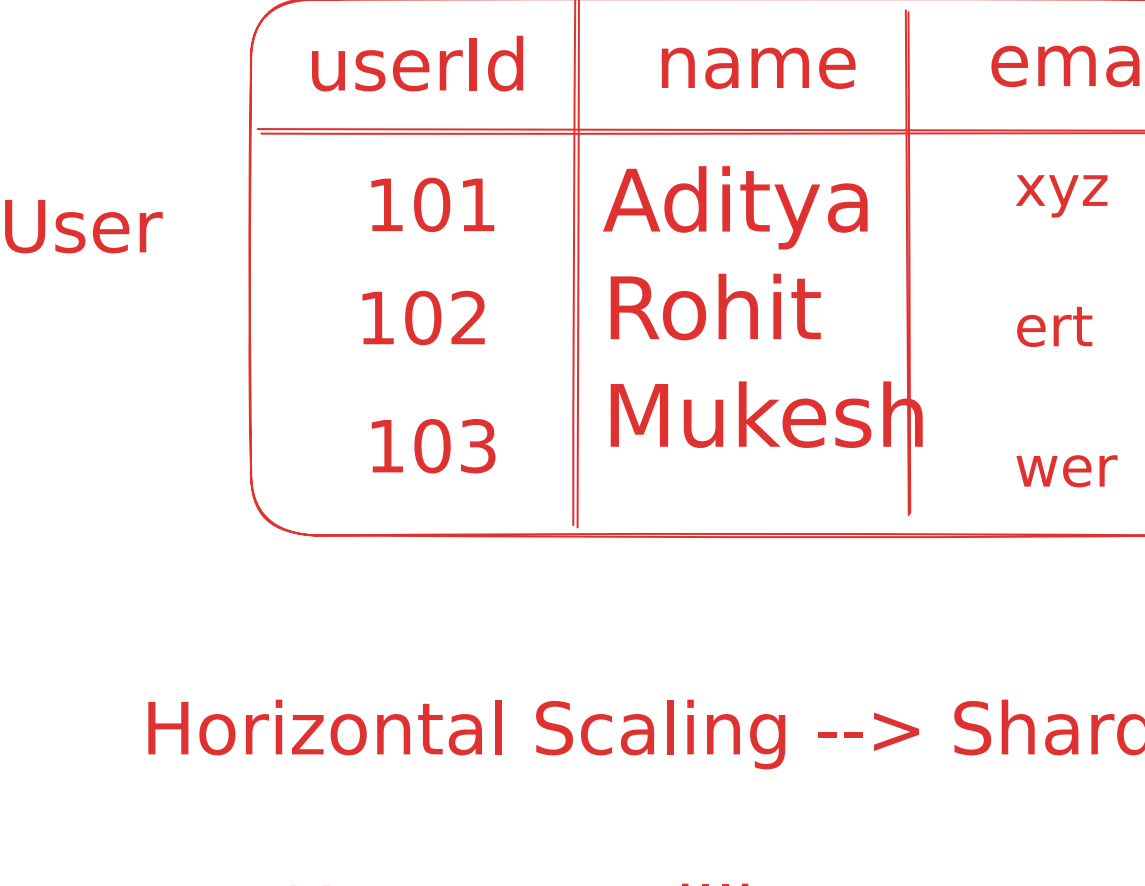


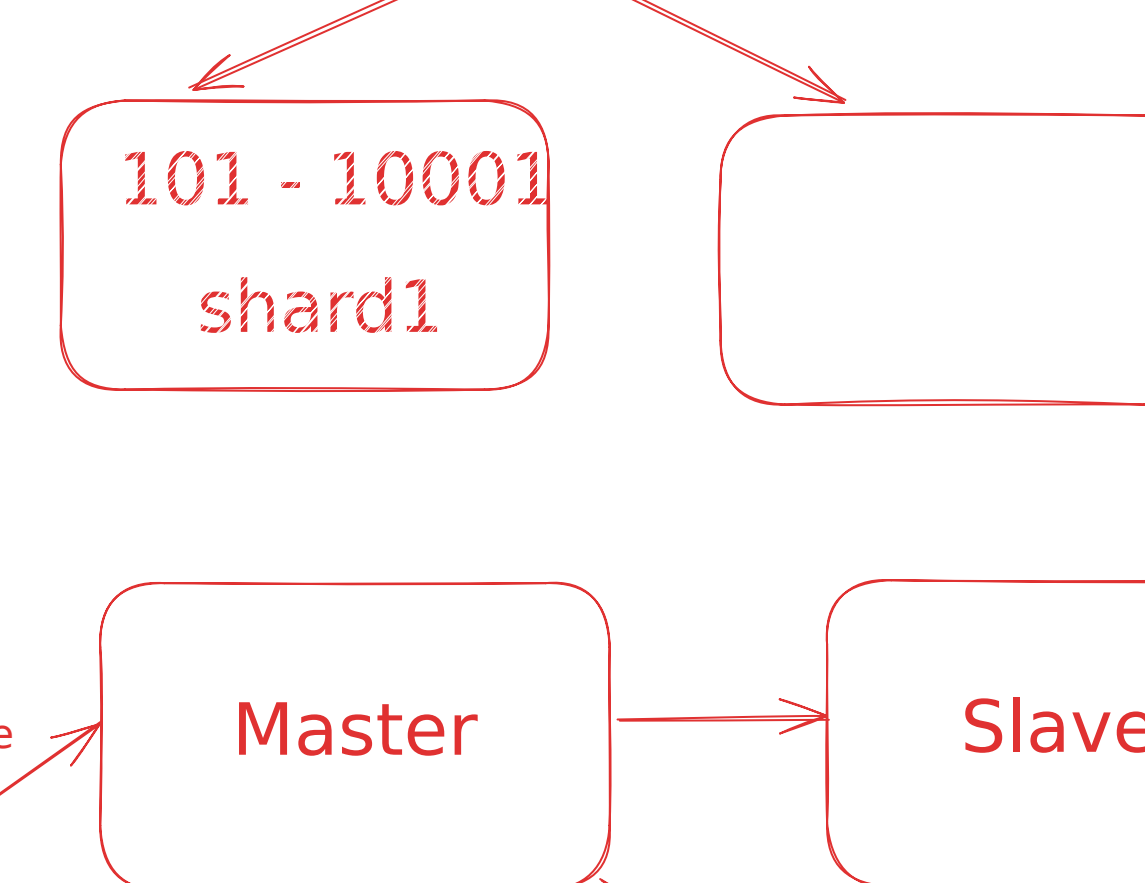
SQL / Relational Architecture



Master-Slave Architecture



Horizontal Scaling --> Sharding



Views. --> Virtual Tables

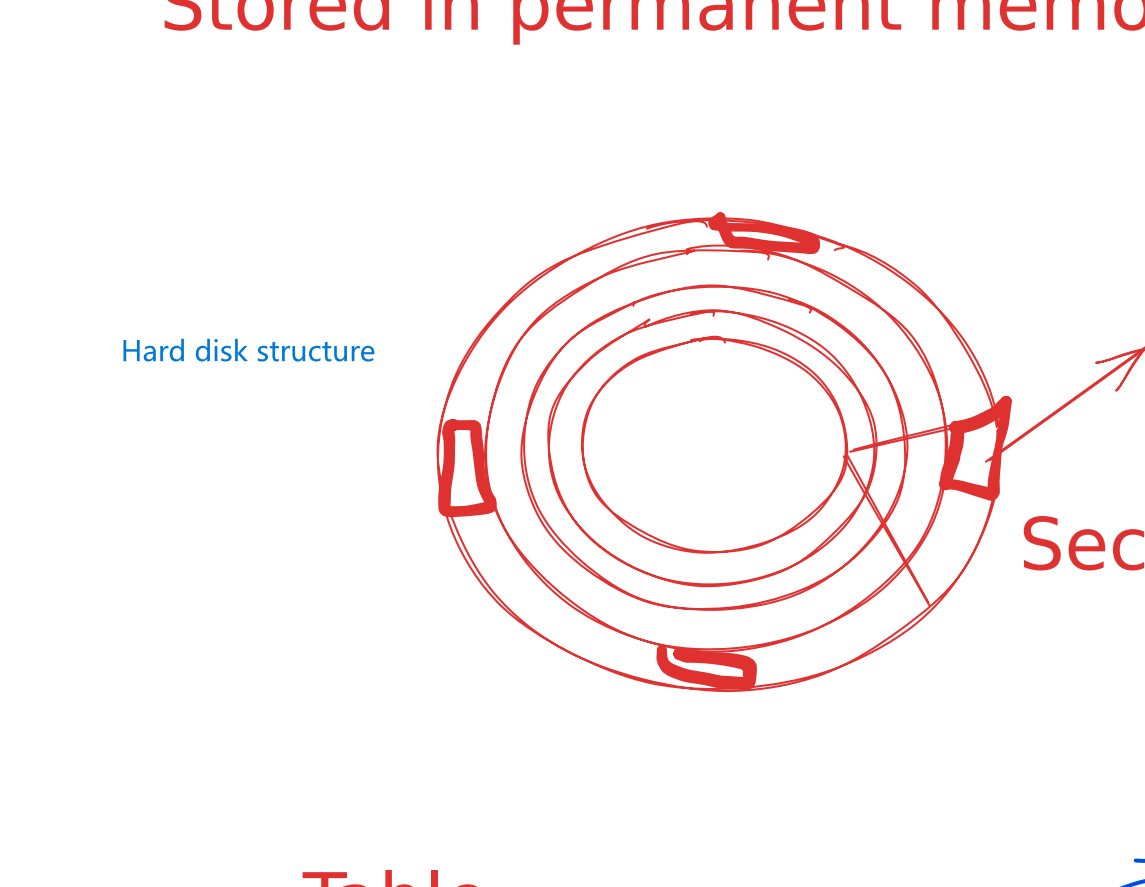


Relational DB. --> Rows and Columns

Logical Arrangement

Stored not in Volatile memory like RAM

Stored in permanent memory (Disc)



Data Block

DBMS

Table

Data Page

Headers

Data Records

offset

Page Number, checksum

Array of indexed rows

1 Million Rows

row = 32 bytes

1 Data Page = ~251 rows.

Total page. = 3984 pages

Data Page

Headers

Data Records

offset

1 Data Block --> 8 KB - 64 KB

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.

Data Page

hard Disk

Select * from user where userld = 101;

Data Block 1 - page 1, Page 2

Data Block 2 - page 3, page 4

Data Block 3

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

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

DBMS has a mapping table (User)

Data page 1 --> Data block 1

Data page 2 --> Data block 1

Data page 3 --> Data block 2

....

Data page 3984 --> Data block 1992

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

for B+ trees

Indexing

Clustered Indexing

Non-Clustered Indexing

B Trees

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

root

Node

M-Way tree

M = 3

keys = M-1 = 2

BST --> Binary Search Tree (2)

TST --> Ternary Search Tree (3)

M- way Trees --> M children (M)

root

Node

M-Way tree

M = 3

k = M-1 (3-1) = 2

no. of keys

B Trees

4 5

Node

M-Way tree

M = 3

keys = M-1 = 2

Root node

sorted order (m-order)

B-Tree Split up

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

9 11

9 11

3 7

10 19

5 9

3 4

7 8

B+ Trees

leaf nodes connected

Intermediate nodes also store parent left value

B-Tree B+ Tree Code

12 19

5 10

12 13

3 4

5 7

Indexing

B+ Trees

PK user

1 Million records

M_way (m ?)

B+ tree of userld

17 25

1 6

1 19

25 30

Data page 1

250 values

19, Aditya, xyz

25, Rohit, ert

30, mukesh, wer

DBMS

data page --> Data block

mapping

asymptotically B and B+ tree 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

19, Aditya, xyz

30, Rohit, ert

25, mukesh, wer

19 25 30

Data Page

in data records data is inserted in same order as it is 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 offset are mapped with their corresponding data in the data records of the page

Indexing

Clustered

Non Clustered

Leaf nodes of B+ Trees store complete row information

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

offsets plays the role

Indexing : Clustered vs Non-Clustered

--> There can be only 1 clustered index per Table.

--> There can be multiple non clustered index per table.

--> What is the issue to index each column??

--> It will require a lot of space.

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

NO-SQL DB

Mongo Db --> B+ Trees

Cassandra DB --> LSM Tree

for indexing

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