1. Good Evening

2. Lecture begins at 9:05

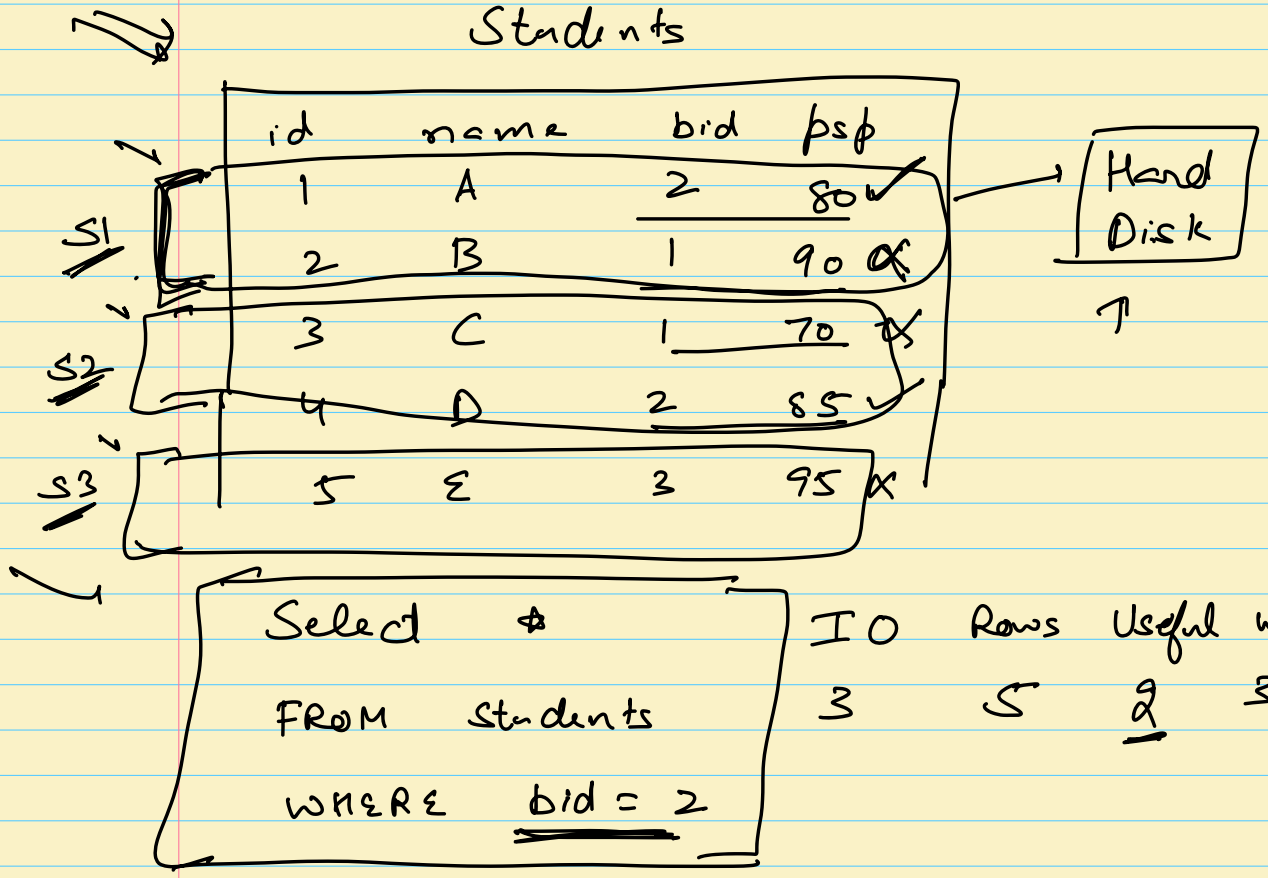
3. Topic – Indexes & Backlog

Agenda

1. Indexes
2. Backlogs → [Data Types. / Builtin Fns. / Serializable → ✓ / Deadlocks]

Intro of Indexes.

Consider the table.




```
Select *
FROM students
WHERE bid = 2
```

IO Rows Useful Wasted

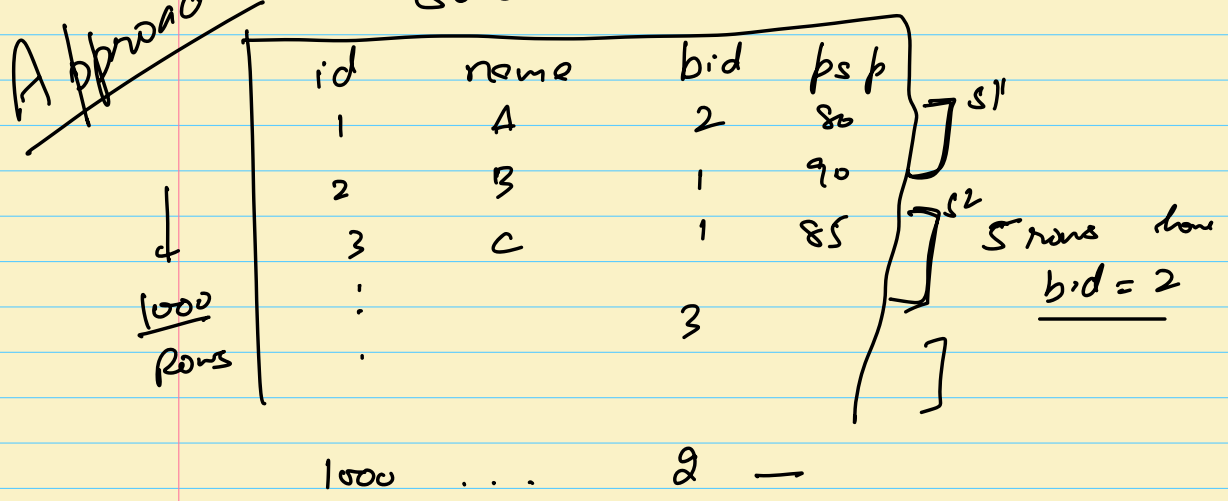
3 5 2 3

CPU — 3 GHz $[3 \times 10^9$ ops/second$]$

Registers >> Cache >> RAM <-> Hard Disk

1. IO operation will bring data from hard-disk to RAM, segment by segment.
2. Once the data is in RAM, it is analysed by CPU & rows are selected/rejected to prepare result.

Approach 1



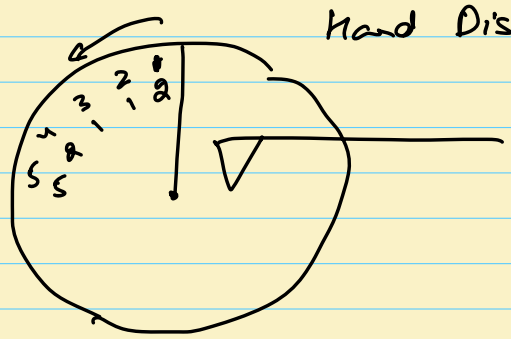
IO	Rows	Useful	Wasted
500	1000	5	995

Query :

```
Select  *
FROM    students
WHERE   id = 5
```

Fact → Data of a table is sorted by PK on the hard-disk

	id	name	bid	psp	
	1	A	2	—	S1
	2	B	1	—	S1
1000 Rows	3	C	1	—	S2
	4	D	2	—	S2
	5	E	5	—	S3

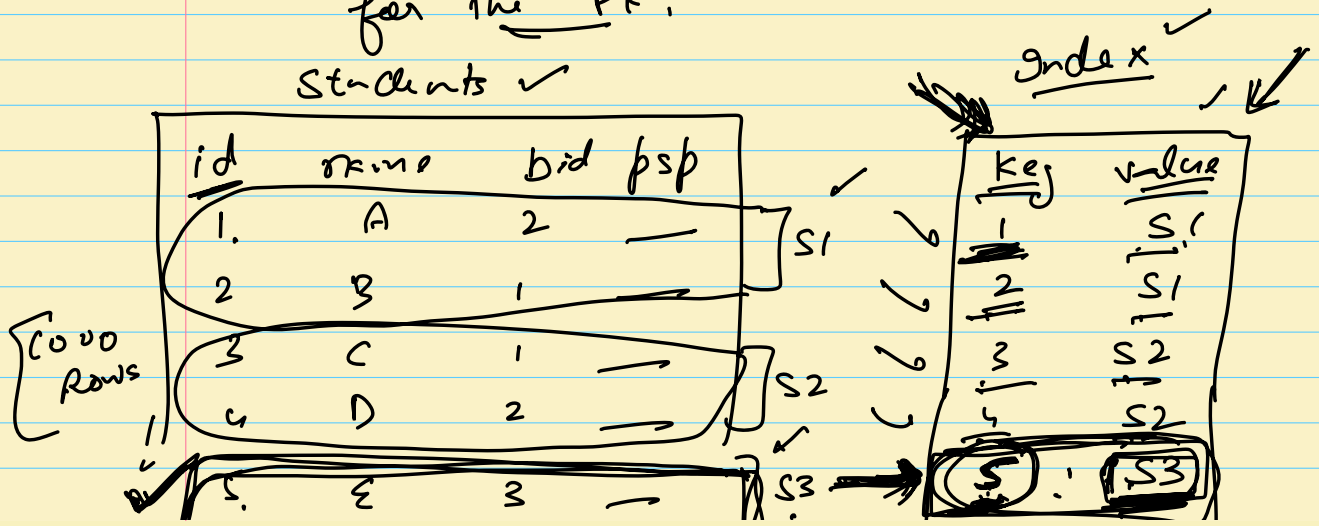


IO	Rows	Useful	Wasted
3	6	1	5

SUMMARY → Where clause on PK will always work faster so data is sorted by PK & PK is unique also.. We will stop reading future segments when we will find desired row.

Approach 3 : Lookup Table / Index

→ For every table, MySQL creates a lookup table or index [stored in RAM as well as hard-disk] for the PK.



6. F 2

| 6 | 53 |

DSA = BTrees

```
Select *
FROM Students
WHERE id = 5
```

1. Check lookup for segment.
2. Read segment
3. Process.

IO	Rows	Useful	Wasted
1	2	1	1

Question : How fast will the search be in lookup table ?

DSA used to implement Index = BTree

Case 1	Case 2	Case 3
pid	id — sorted	id — sorted Index

```
Select *
FROM Students
WHERE bid = 2
```

Students

id	name	bid	psp	
1	A	2	—	S1
2	B	1	—	
3	C	1	—	S2
4	D	3	—	
5	E	2	—	S3
6	F	1	—	
7	G	3	—	S4

* PK has index by default.
* We can create indexes for other columns as required.

key (bid)	val (List of sysnos)
1	: S1, S2, S3
2	: S1, S3
3	: S2, S4

BTree

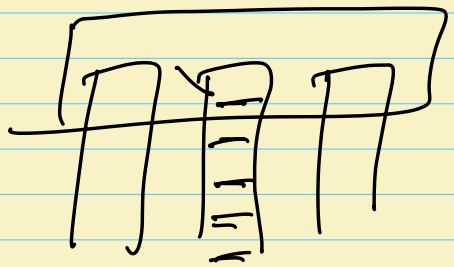
IO	Rows	Useful	wasted.
2	4	2	2

- Pros & Cons
- Guidelines
- Composite Indexes

BREAK = 10:18 to 10:25

1. Demo — show, CREATE, DROP, Explain ✓

2. DSA for index ✓
3. Pros & Cons
4. Guidelines.
5. Indexes on String
6. Composite Indexes → (01) (02)



DSA FOR Index

key	value
1	S1
2	S1
—	-
—	

Map

Java
- HashMap
- TreeMap

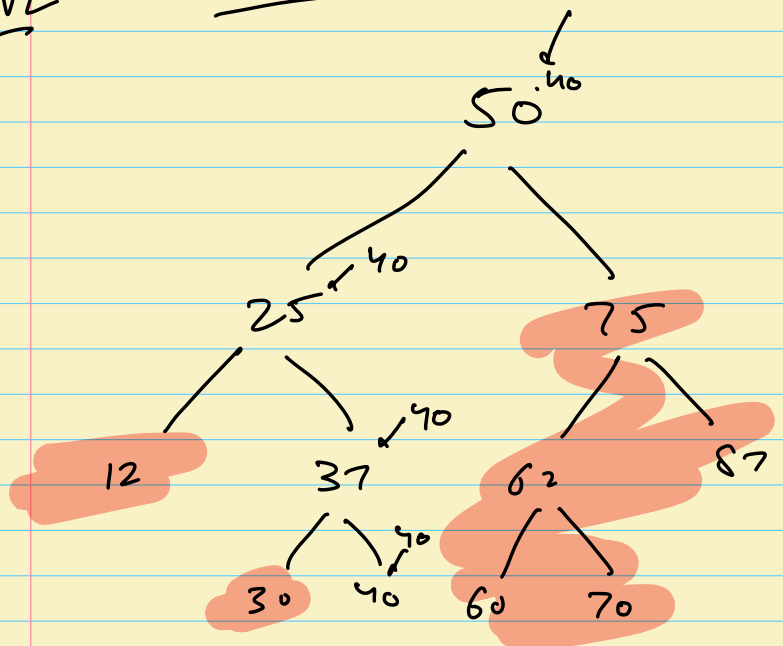
C++
- Unordered_Map → HashTable
- Ordered_Map → BST

DSA

HashTable → Avg constant $O(1)$

Balanced BST → $O(\log n)$

- AVL
- Red-Black



$\log_2 n$

Faster?

$\log_2 n$ << $\log_{10} n$

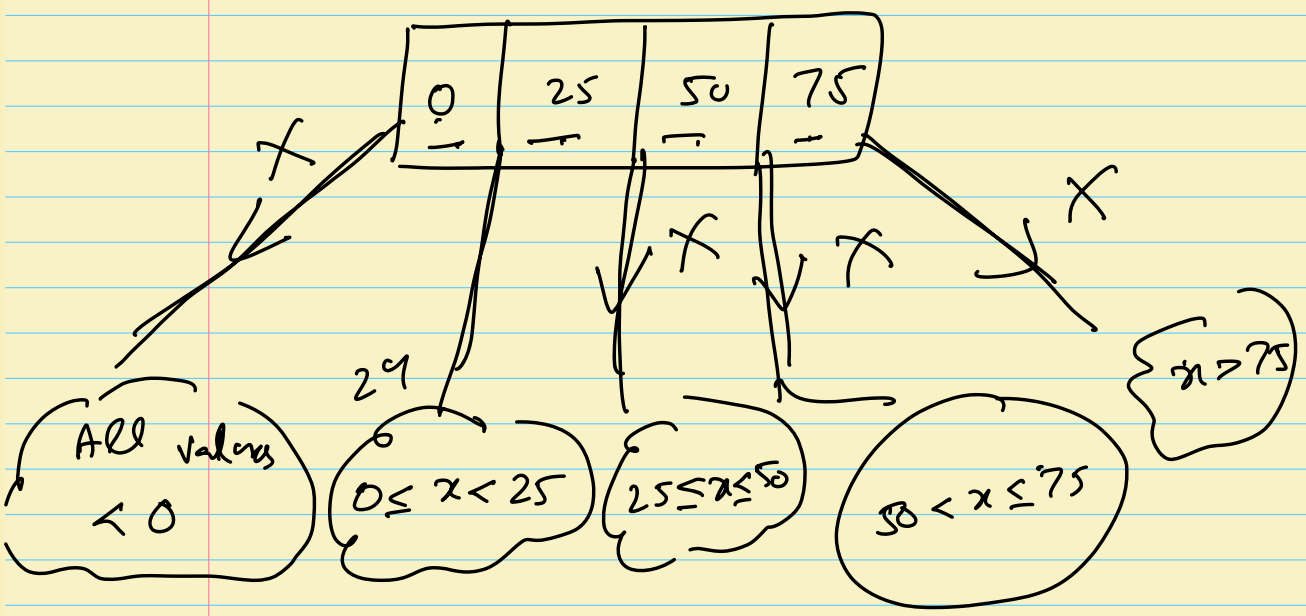
$\log_2 1024$

10 operations.

$\log_{10} 1024$

3 comparisons.

B Tree



$T_{pick} = \log_{26} n$

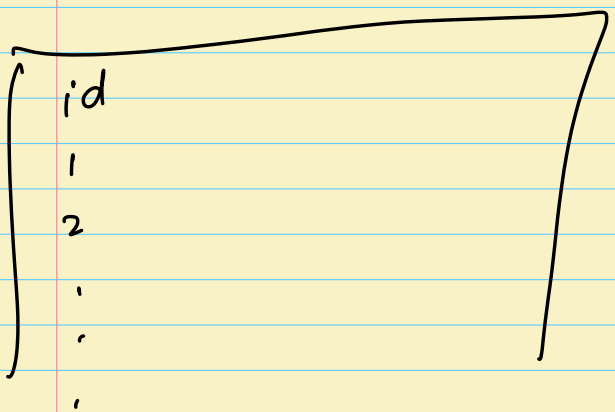
Algorithm = $\log_{degree} n$

B Trees { degree can be as high as 2000 }

$\log_{2000} n$

8×10^9 rows

$\log_{2000} n$



8×10^9 rows

$\log_{2000} 8 \times 10^9$

= 3 comparisons

1. Indexes uses BTrees.
2. BTrees are similar to BSTs
3. The Performance of BTree is $\log_{degree} n$

⟹ How to implement using BTrees for DBMS

✡ PROS & CONS of Indexes.

PROS

1. Fast search ° ° °
of optimised reads.
We read only relevant segments, we decrease the number of non-useful rows read.

CONS

1. Extra space per lookup.
2. INSERT, Delete Update might become slower.

Students

id	name	bid	psp	
1	A	2	80	S1
2	B	1	90	
3	C	1	75	S2
4	D	2	80	
5	E	3	90	S3
6	F	1	75	
7	G	2	80	

Idx_id

id	segment
1	S1
2	S1
3	S2
4	S2
5	S3
6	S3

Idx_Name

n	segment
A	S1
B	S1
C	S2
D	S2
E	S3
F	S3

Idx_bid

bid	segment
2	S1, S2
1	S1, S2, S3
3	S3

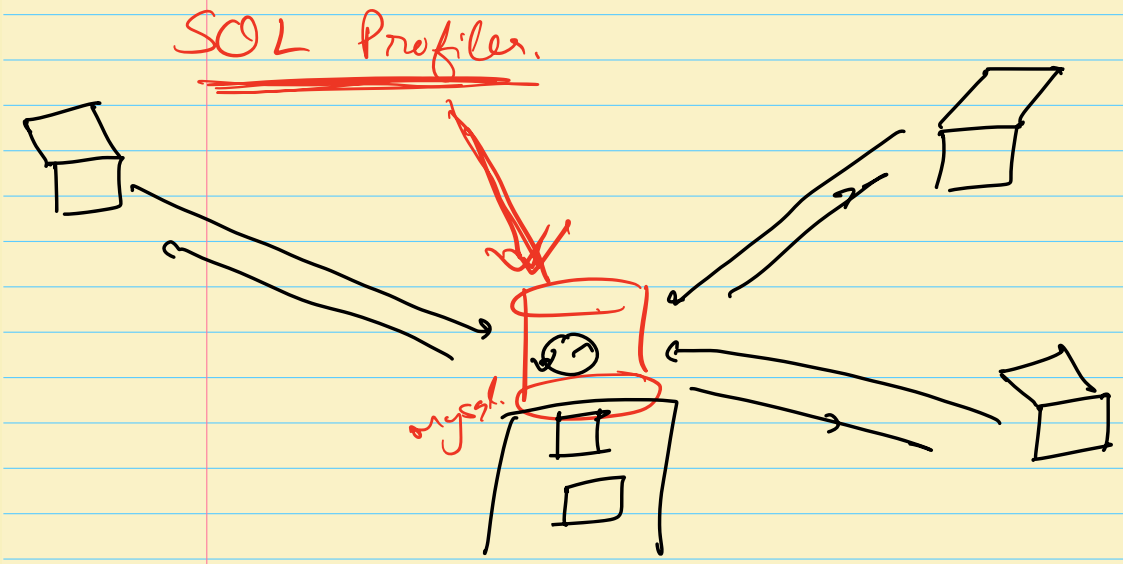
Idx_psp

psp	segment

GUIDELINES

1. Don't create index while creating tables [let the PK index get created]

2. Analyse the traffic on db.



3. Create index for columns, which are frequently used by where clause [as per data logged by SQL Profiler].

Indexes on String

Students

id	name	bid	psp	
1	Sumeet	—	—	S1.
2	Mayank	—	—	
3	Naman	—	—	S2
4	Megha	—	—	
5	Smriti	—	—	S3.
6	Nayan	—	—	
7	Mukesh	—	—	S4.
8	Sudheer.	—	—	

Sudheer

Case 1 : No index.

IO	Rows	Useful	Wasted.
4	8	1	7

Case 2 : Index on name column

IO	Rows	Useful	wasted.
1	2	1	1

Sumeet	S1
Mayank	S1
Naman.	S2
Megha	S2
Smriti	S3
Nayan	S3
Mukesh	S4

| Sudheer S4 |

Case 3: Index on 1st char of name.

IO	Rows	Useful	Wasted.
3	6	1.	5

S	S1, S3, S4
M	S1, S2, S4
N	S2, S3

Case 3: Index of 1st 2 char

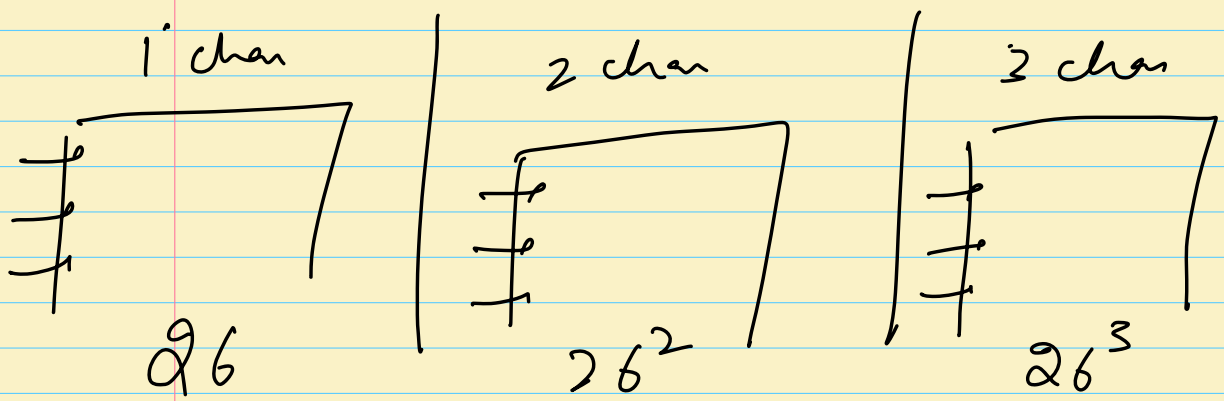
IO	Rows	Useful	Wasted
2	4	1	3

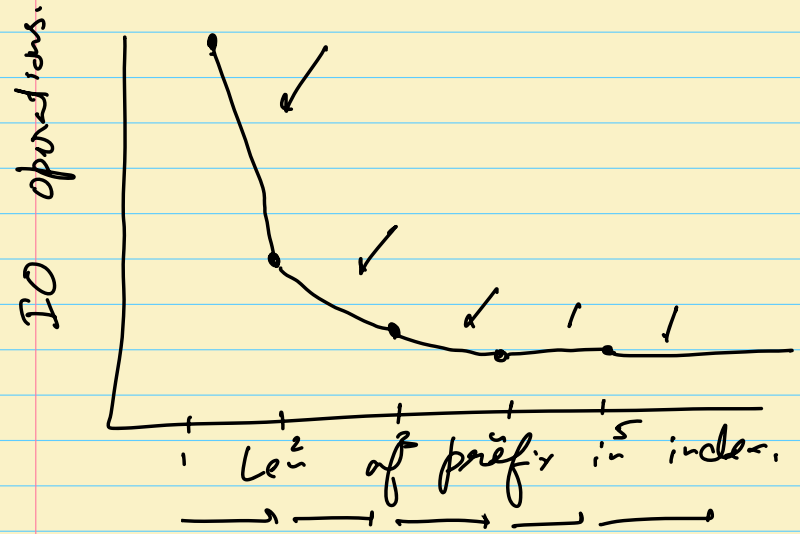
Su	S1, S4
Ma	S1
Na	S2, S3
Me	S2
Sm	S3
Mu	S4

Case 4: Index on 1st 3 chars

IO	Raw	Useful	wasted
1	2	1	1

Sam — S1
May — S1
Nam — S2
Maj — S2
Sman — S3
Nay — S3
Mak — S4
Snd — S4



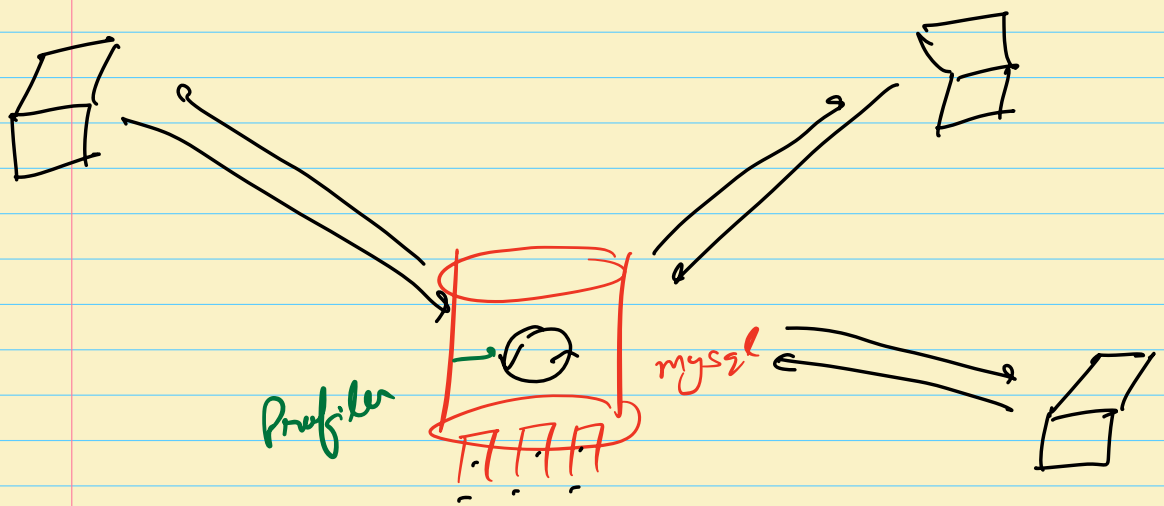


Backlog

+ PROCEDURES + UDFs.

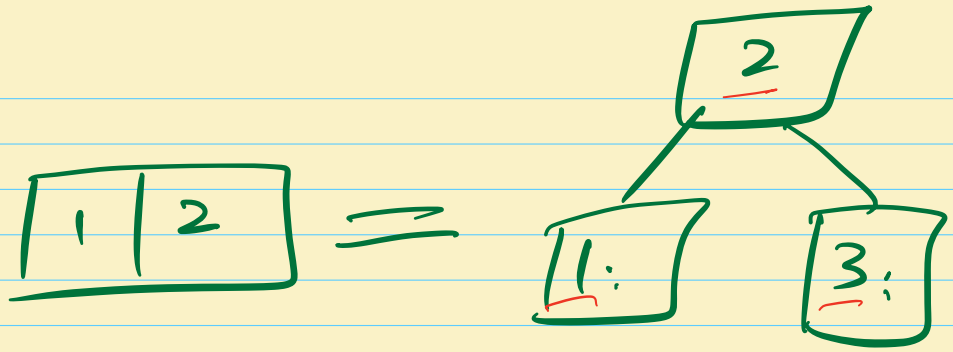
1. Data Types
2. Builtin Fns.
3. SERIALIZABLE + Demo + DeadLocks
4. COMPOSITE Indexes
5. Full-text SEARCH

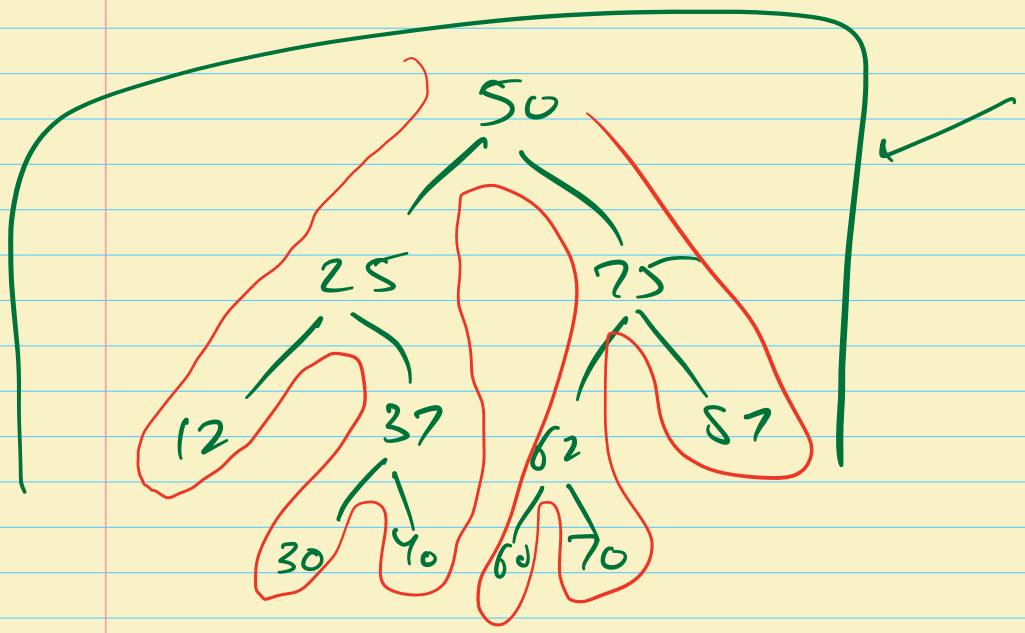
SUB-QUERY + VIEWS

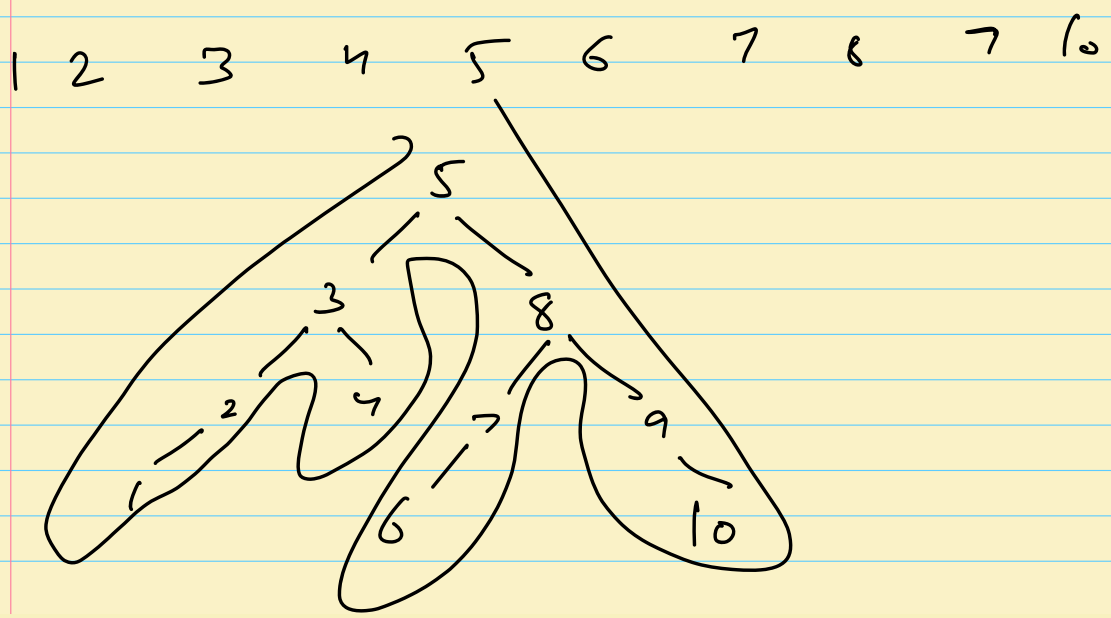


Robert Vieira

- Beginning MS SQL Server 2005
- MS SQL Server 2012 for Advance Programmers.


1 | 2
2
1:
3:


50
25
75
12
37
62
87
30
40
60
70


1 2 3 4 5 6 7 8 9 10
5
3
8
2
4
7
9
1
6
10