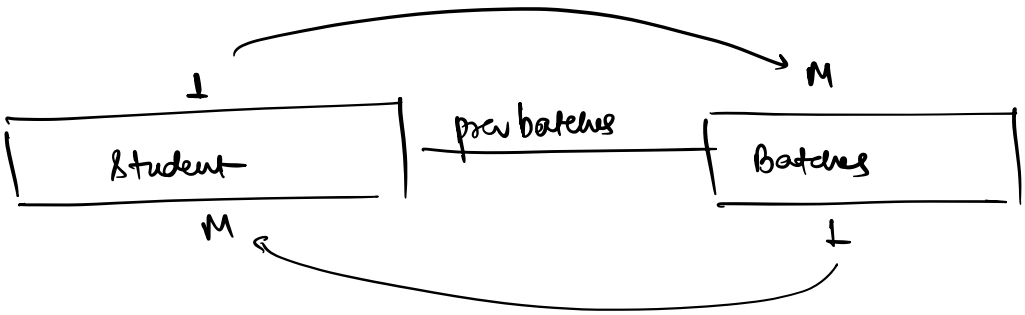


M:M

Students

id	Name	phone
1	Sudhi	1234

pk

Batches

id	name	startdate
1	Feb22	Feb 22

pk

fk fk

id	student_id	batch_id

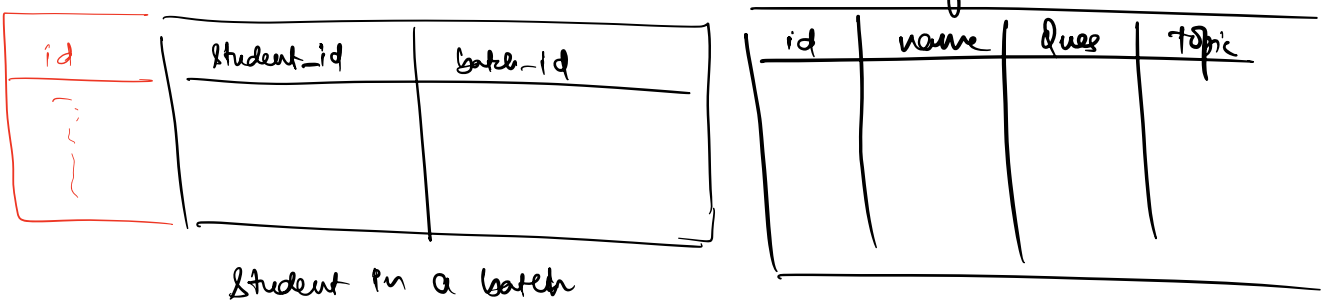
Two options for pk of this table:

do this always, unless there is a strong reason, not to do

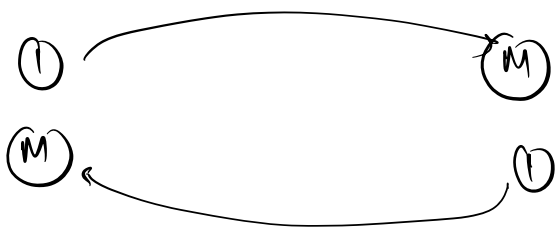
1) Combo of both fk, can be considered as pk (student_id, batch_id)

2) add a third column, called id, and generate pk

In a scenario, where there is a relationship of the mapping table, with something else

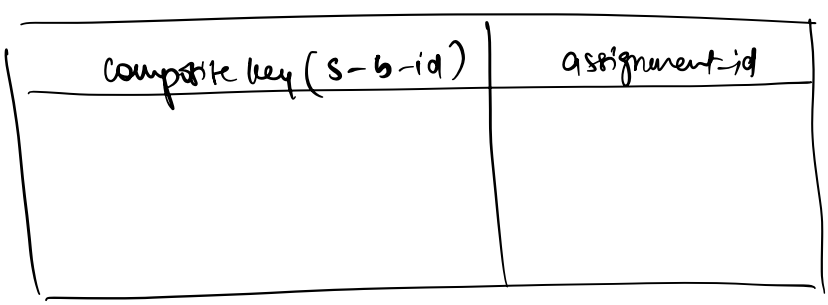


Student in a batch



M:M

1) Composite => pk (student_id, batch_id)



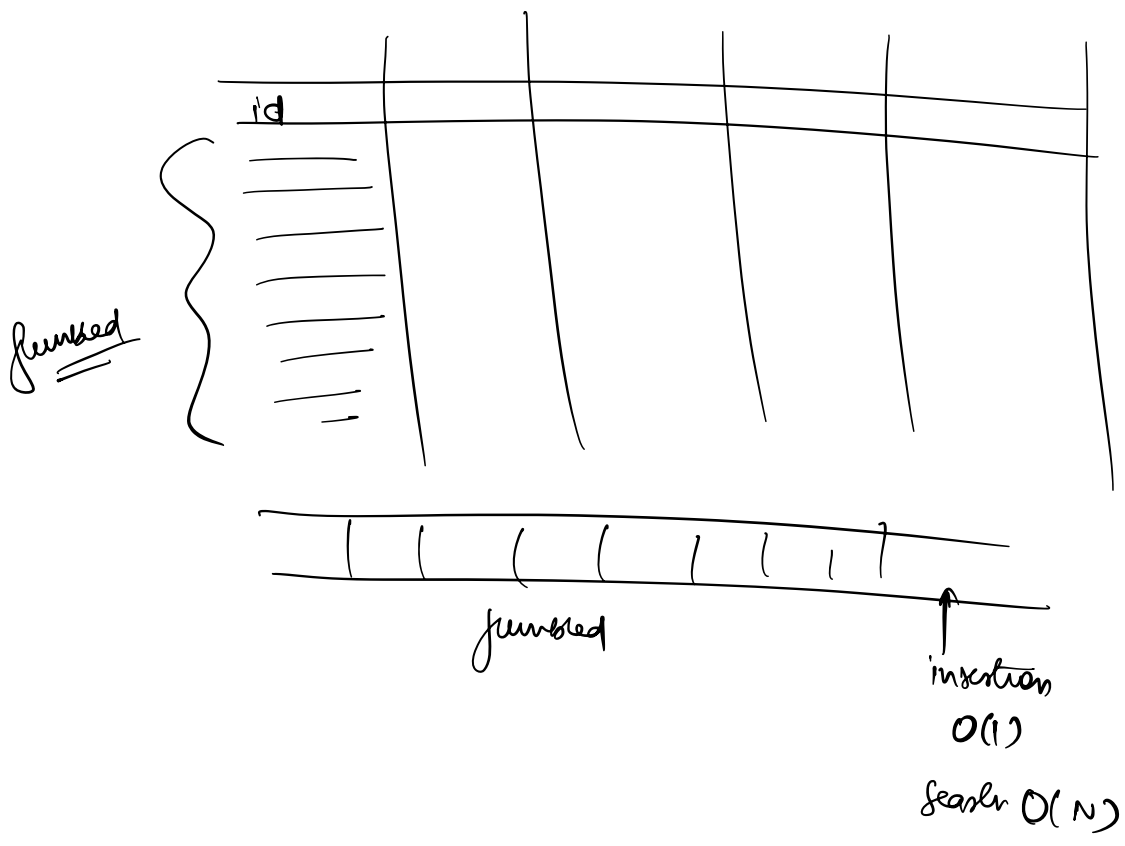
ii) Id column

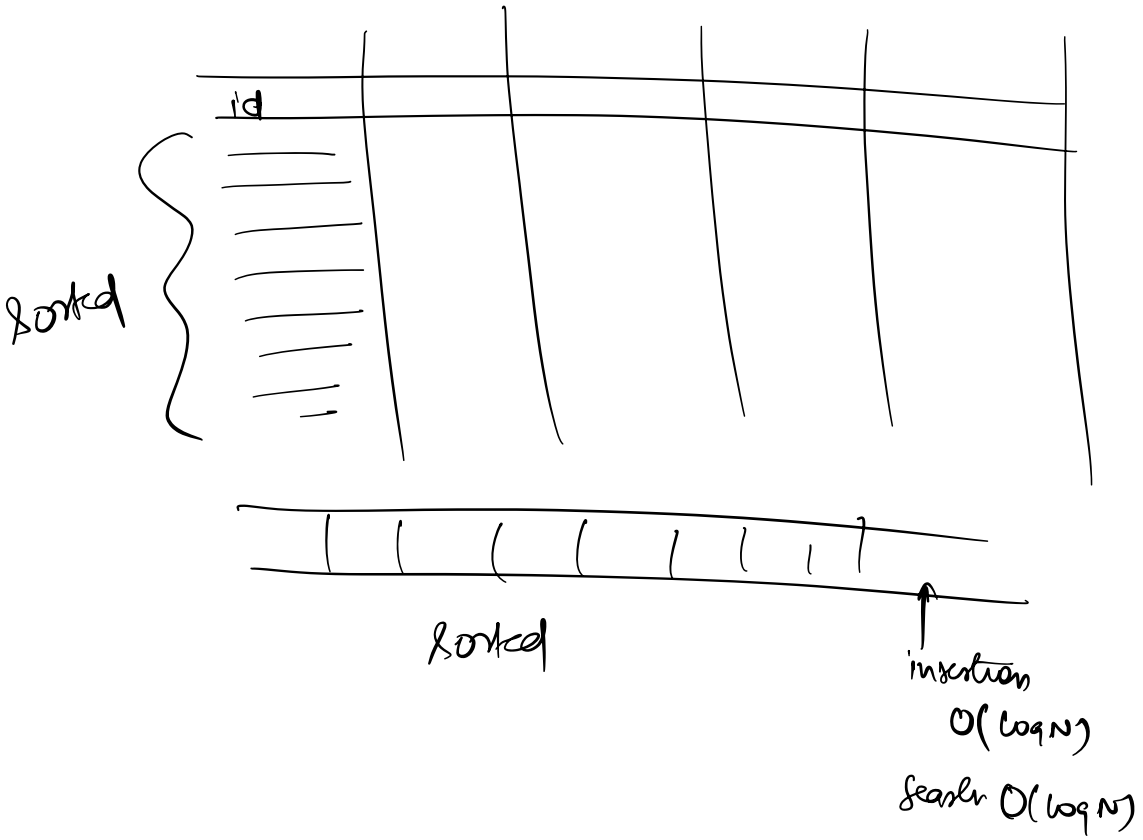
=> why prefer composite pk ? Student batch

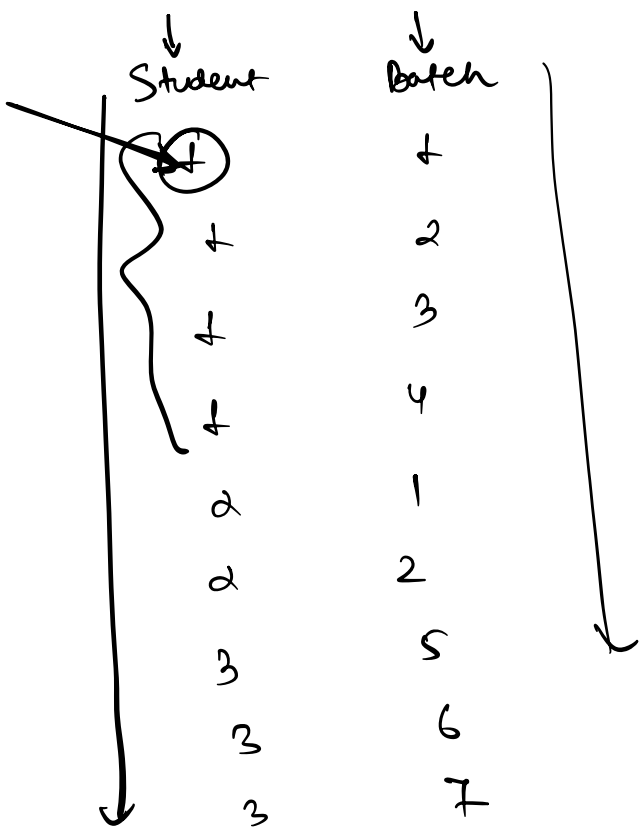
If the pk composite is (s_id, b_id) then if indexed the data will be first sorted by student[s_id].

Linear Search => O(N)

Binary Search => O(log N)



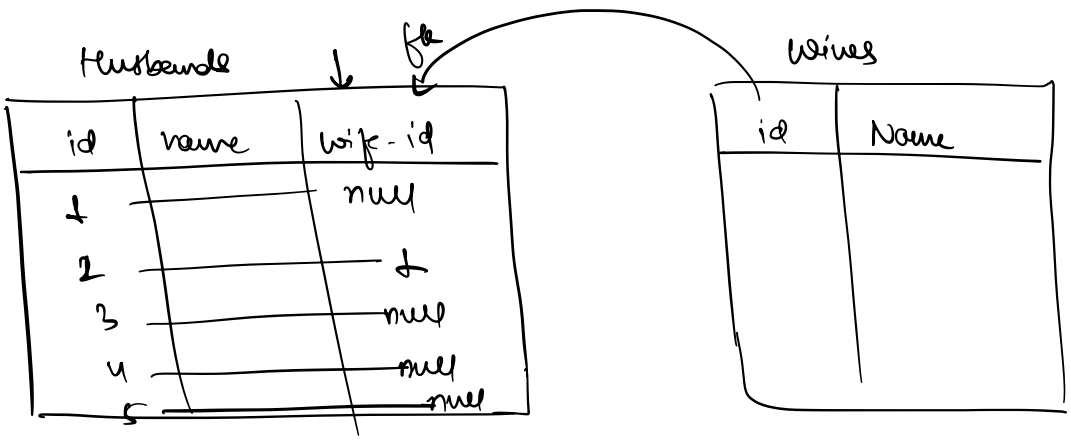

id
Sorted
Sorted
insertion
O(logN)
Search O(logN)


Student
Batch
1 1
1 2
1 3
1 4
2 1
2 2
3 5
3 6
3 7

⇒ Important points:-

⇒ 1:1 | 1:M | M:1 :-

Eligible Husband / Bachelor wife

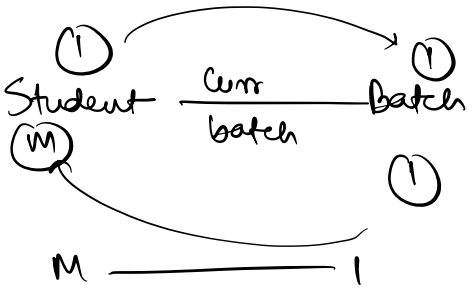


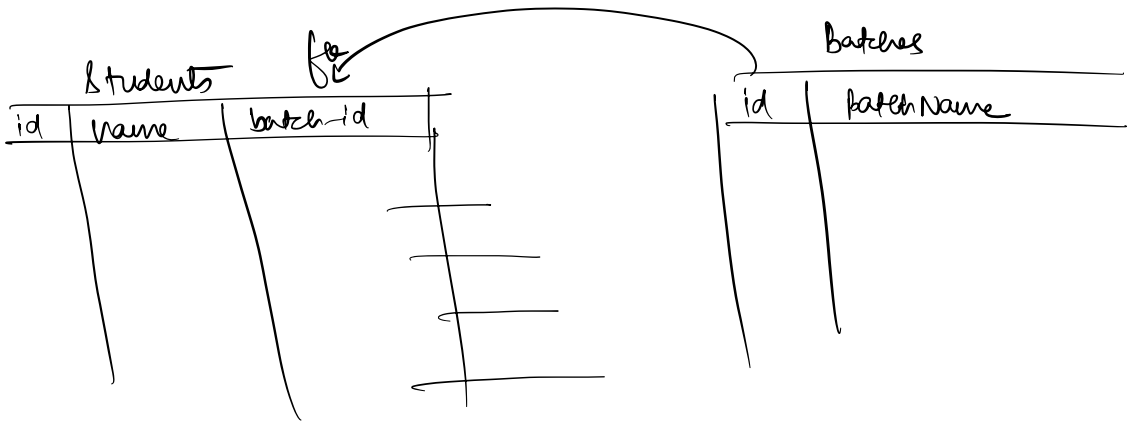
5 million entries ⇒ husband table
2 million got a wife

3 million cells as null ⇒ huge space wastage.

This is an example where relationship table can have sparsed entries as fk.

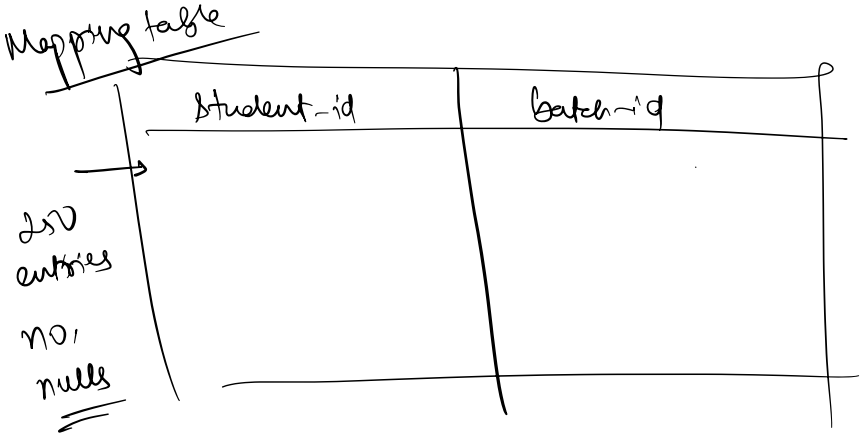
⇒ Create a mapping table to save space





1500 → 250 [allocated batches]

No of null batch_id : 1250



husbands

id	name	wife_id
1	ABC	1

wives

id	name
1	XYZ

id	name	wife_id	marriage_date	date of engagement	first

↓

dont store all info in husband table, because a lot of attributes are dependent on the mapping of husband and wife

Marriage Date

id	husband_id	wife_id	marriage Date	engagement date	money spent in marr	cri for marr

SCENARIO / CARDINALITY	Normal	Sparse	List of attributes	
1 : 1	PK of one table to another side as FK	Mapping table	Mapping table	
1 : M or M : 1	PK of one table to another side as FK	Mapping table	Mapping table	
M : M	Mapping table	Mapping table	Mapping table	

⇒ Types of DataTypes ⇒ [MySQL]

i) String

ii) Integers

iii) float

iv) Boolean

v) Enum

vi) Date / Time

vii) JSON

viii) BLOB

⇒ String Data types:-

- char
- varchar
- & text

— char(x)

- Strings of fixed lengths
- x can be 0 to 255

ex ⇒ char(4) → data

char(4)

⇒ abcd ✓

⇒ pincode ⇒ abc ⇒ "abc␣" ✓

⇒ phone no ⇒ ab ⇒ "ab␣␣" ✓

⇒ card no. ⇒ a ⇒ "a␣␣␣" ✓

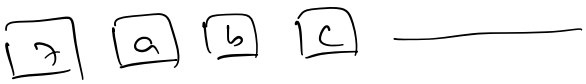
⇒ roll no. ⇒ abcde ⇒ throw an error or, "abcd"

⇒ country currency ⇒ 3 letters - INR, USD, eh ——-

- varchar(x)
  - String of variable length
    - x ⇒ 0 to 65535
  - Initial 1 or 2 bytes are kept for maintaining length of the String.

ex ⇒ "ab" ⇒ 2 ab

"abcd123" ⇒ 7 a b c d 123

⇒ 

"abcdef ... = ... 350 characters"

⇒ 

⇒ name, email, address, tweets

- text [text can't be indexed]

TINY TEXT	TEXT	MEDIUM TEXT	LONG TEXT
255B	64KB	16MB	4GB.

ex ⇒ Blogpost, Articles, Research papers

II) Integers Data type :-

TINYINT ⟶ 1B ⇒ -128 to 127

UNSIGNED TINYINT → 1B ⇒ 0 to 255

SMALLINT ⟶ 2B → [-32k, 32k]

MEDIUMINT ⟶ 3B [-8M, 8M]

INT ⟶ 4B [-2B, 2B]

BIGINT ⟶ 8B [-9ZB, 9ZB]

III) Float types:

a) Decimal (p, s) → length after decimal

↓

total length

ex ⇒ 320.04 ⇒ p = 5

s = 2

ex ⇒ 1.9 - 0.9 == 1 ⇒ true

[1.8999999 - 0.90000001 == 1

false