Aggregate functions.

Agenda.

aggregate fns
- group by
- having
- having vs where

functions.
- count()
- max()
- min()
- avg()
- sum()

→ till now whatever SQL queries we were writing were to get few of the records of a table / tables

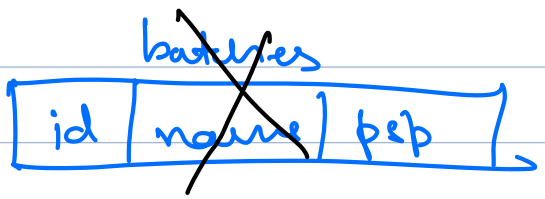
→ find the students where id = 2
→ find " " batch 1, 2, 5

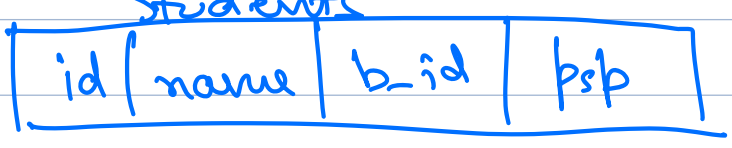
→ sometimes, instead of getting a few of the entities, we have to get some analysis out of some entities.

↓

get data that is formed as a combination of rows.

Q- Get avg psp of every batch of scaler.





How to solve this code.

↓

go through students table

↓

combine rows of students of a batch.

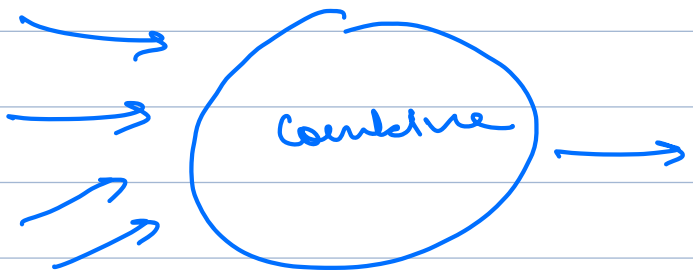
↓

calculate the avg

Aggregate functions

count( ) sum( ) max( ) min( ) avg( )

count(_ _ _ _) → 4



① Count.

Students.

id	name	age	b_id
1	A	20	1
2	B	21	1
3	C	22	null
4	D	23	2

Q Count the students that have a batch?

count(1, 2, 3, 4, 5) → 5

Count → takes a lot of values and combines them into a single value which is equal to count of values of the set.

How to use it.

```
select count(b_id)
from students;
```

→ for loop.

id	name	age	b_id
1	A	20	1
2	B	21	1
3	C	22	null
4	D	23	2

3

count(1, 1, null, 2) ⇒ 3

⇒ All aggregate functions ignore null values.

Q select count of students where age < 23.

id	name	age	b_id
1	A	20	1
2	B	21	1
3	C	22	null
4	D	23	2

```
Select name, age
from student
where age<23;
```

```
select count(age)
from students
where age < 23;
```

count(20, 21, 22) → 3

Q Count of students that have batches & age<23.

```
select count ( b_id )
from students
where age< 23;
```

count(1, 1, null) → 2

→ Aggregation happens at the end.

↓

first (ans) table is created & then aggregate fn works on top of it.

Q

students

id	name	batch_id
1	a	1
2	b	1
3	c	2

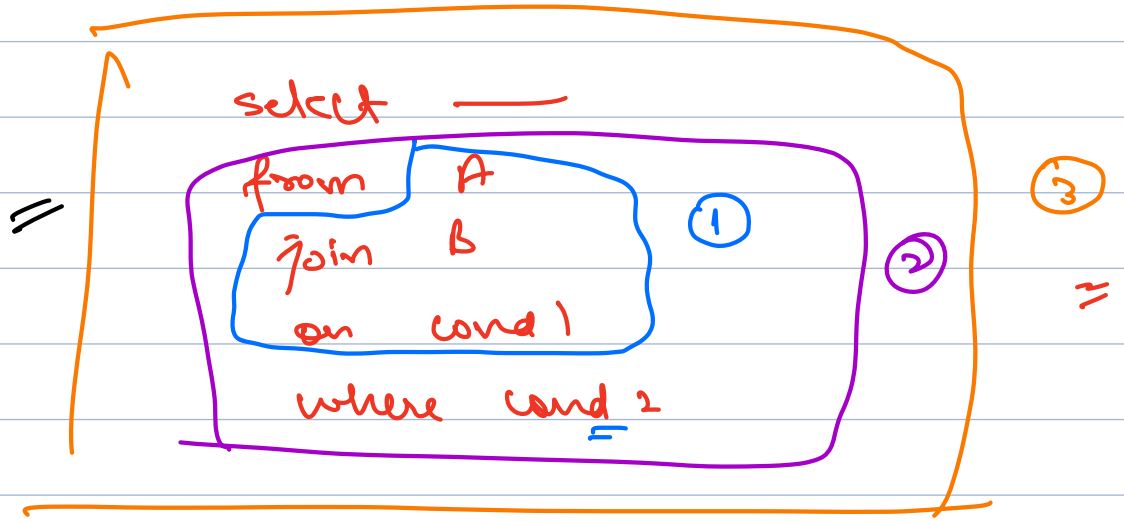
batches

id	name
1	A
2	B

Q Select count of students with batch name of A.

```
select count(s.id)
from students s
join batches b
on s.batch_id = b.id
where b.name = 'A'
```

1	a	1	A
2	b	1	A
3	c	2	B



Code

```
A = [ ]
B = [ ]
ans1 = [ ]

for row1 in A :
    for row2 in B:
        if cond1 is true b/w A & B:                    join
            ans1.add( row1 + row2)

ans2 = [ ]
for row3 in ans1 :
    if row3 satisfies cond2 :                          where
        ans2.add (row3),

count_s_id : 0
for row in ans2:
    if row[s_id] not null :                            aggregate
        count_s_id += 1
```

print (count_s_id) ; select.

on → cond' to join table
where → eliminate extra rows / unwanted rows.

other aggregate functions

→ you can print multiple aggregations
= at the same time.

```
select count (batch_id), sum( psp),
       avg (psp)
from students;
```
→ whatever the table is

from (x) ;

① MAX (- - - -) → maximum value

② MIN (- - - -) → minimum value

↑ should be comparable. eg. int, date etc.

③ Avg ( )

- Avg (1, 2, 3, null) → $\frac{6}{3}$ = 2

students

id	psp
1	1
2	2
3	3
4	null

name

select sum(psp) / count(id) = avg(psp)

(1, 2, 3, null) (1, 2, 3, 4)

6 4

$\frac{6}{4}$

= $\frac{6}{3}$ (1, 2, 3, null)

Q How many students are there? =

```
select count(id)
from students ;
```

Constraints no single colm that will always be non-null.

```
select count(*)
from students;
```

=

```
select count(1)
from student;
```

=

count(s_id)

```
count_s_id : 0
for row in ans2:
    if row[s_id] not null:
        count_s_id += 1
```

```
select count(true)
from students;
```

select count(1) = select count(*)

break till 8:18.

→ Till now in aggregate fns we have a lot of values, that we aggregate / combine to a single value.

Q Get the count of students for every batch.

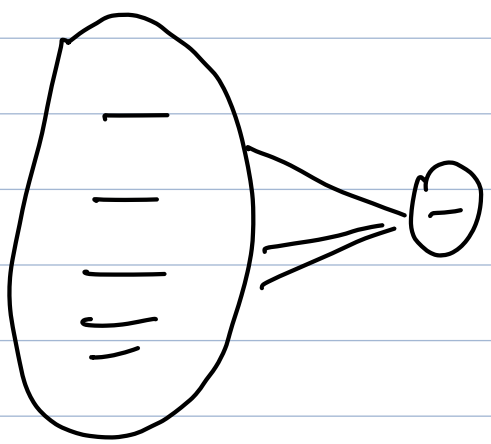
students.

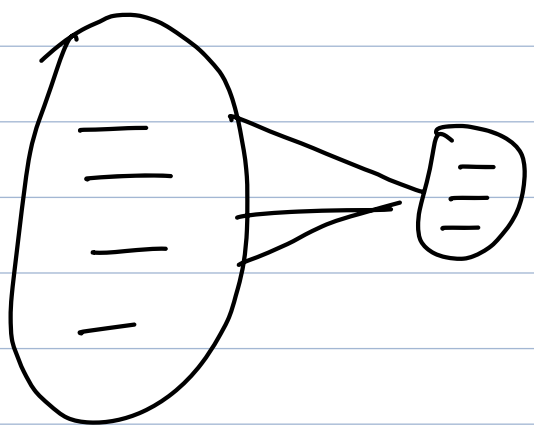
id	name	bid
		1
		1
		1
		1
		2
		2
		3
		3

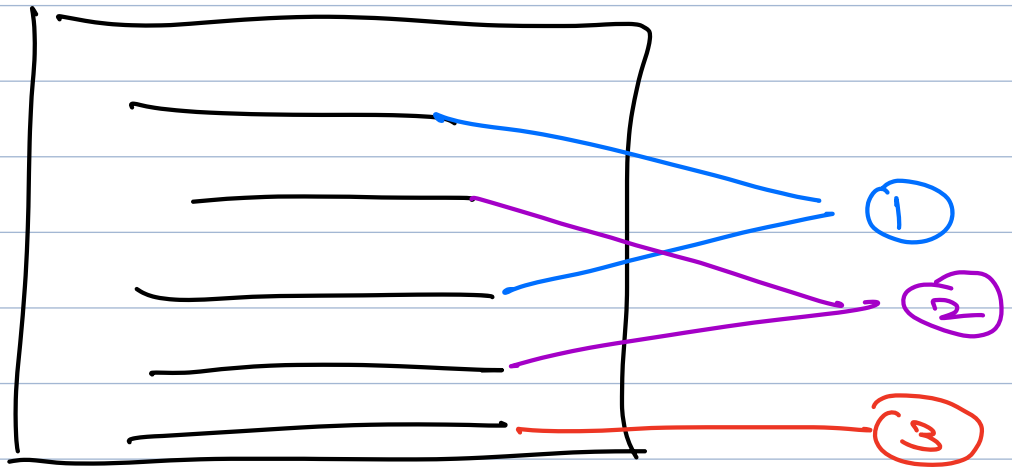
```
select count(*)
from students;
```

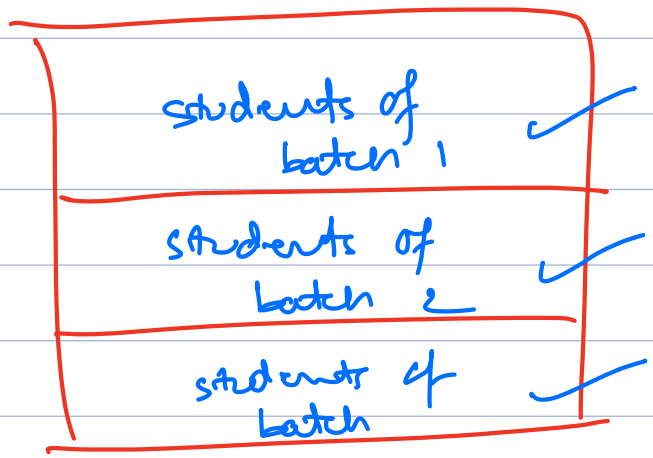
bid	count
1	5
2	2
3	2

→ each of the sets will be independent





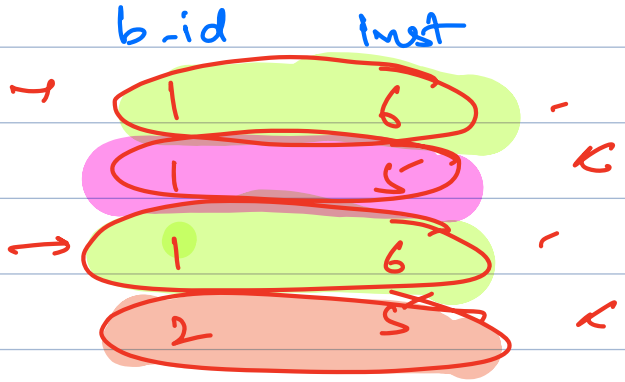




Group by → allows you to break your table into multiple groups so as to be used by aggregate fns.

eg. group by batch_id;
↳ brings all the rows with same batch together.

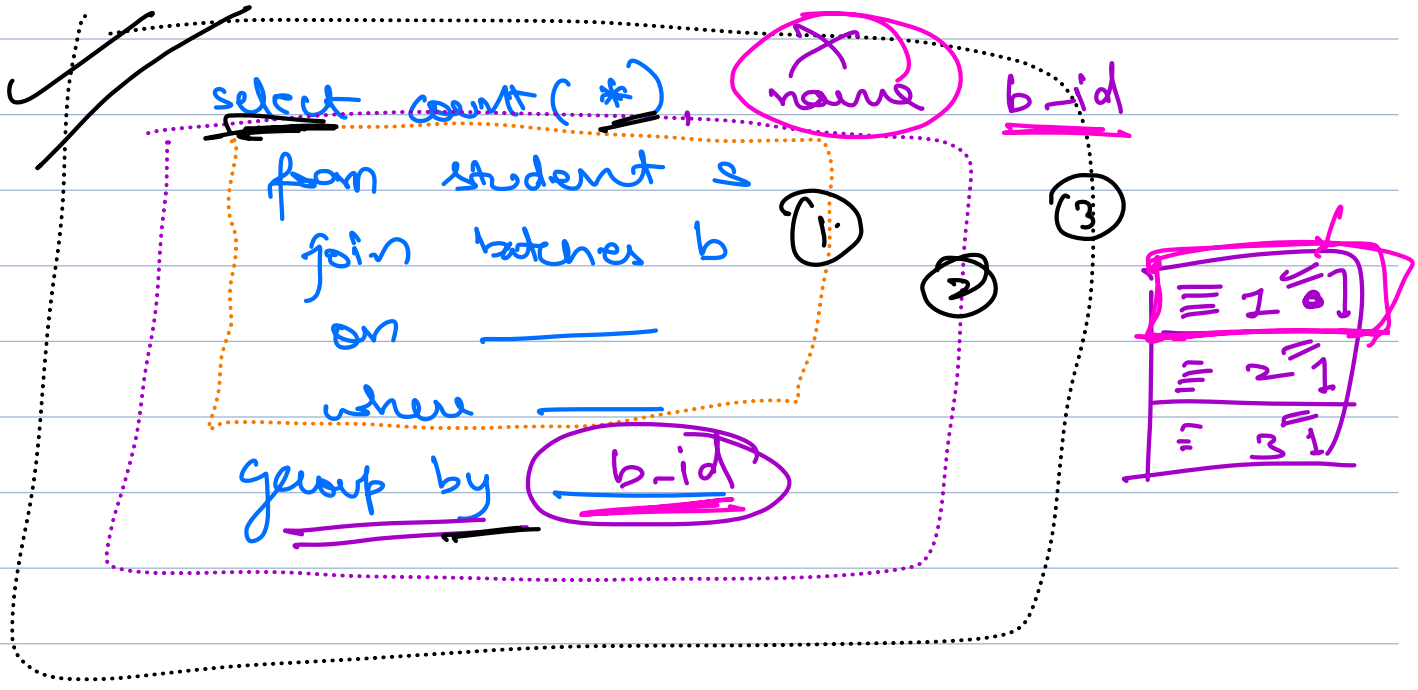
group by batch_id, instructor;




```
select count(*)
from students
group by b_id;
```

→ table

count	b_id
1	1
2	2
2	3



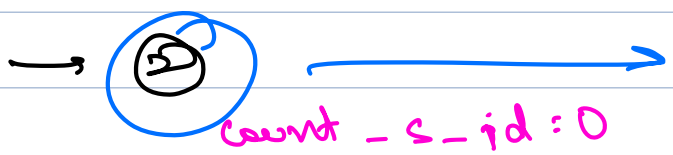
You can only use those col in select that are present in group by.

All values of the same group have one/more values in common that were in the group by statement

```
A = [ ]
B = [ ]
ans1 = [ ]

for row1 in A :
    for row2 in B:                          join
        if cond1 is true b/w A & B:
            ans1.add( row1 + row2)
```

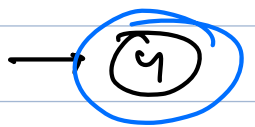

```
ans2 = [ ]
for row3 in ans1 :                          where
    if row3 satisfies cond2 :
        ans2.add (row3);
```



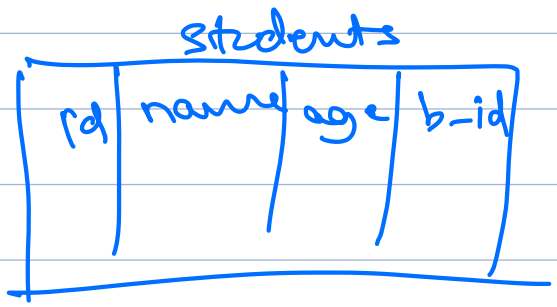
group by.

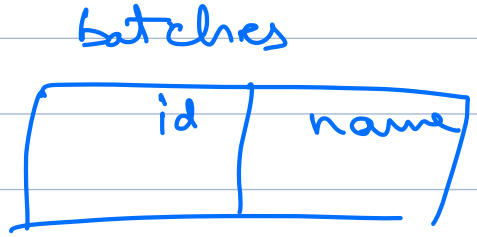
```
count_s_id : 0
for row in ans2:                            aggregate
    if row[ s_id] not null :
        count_s_id += 1
```


```
print (count_s_id) ;
```

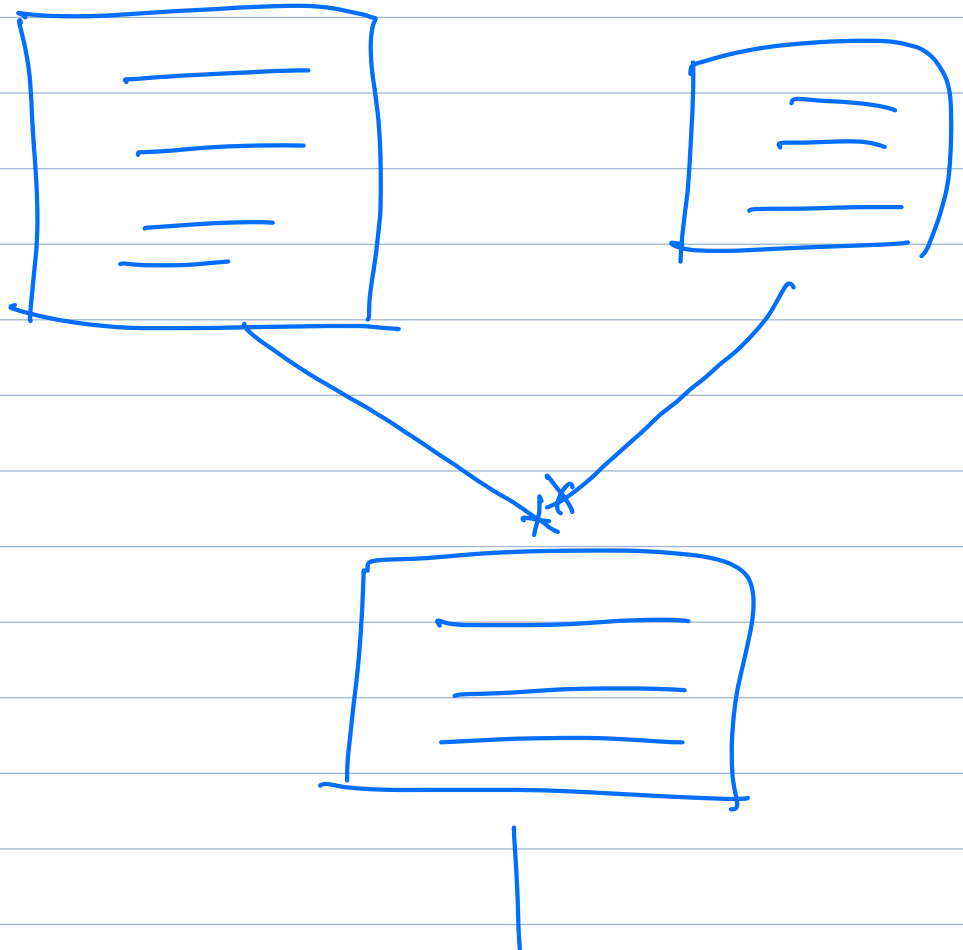


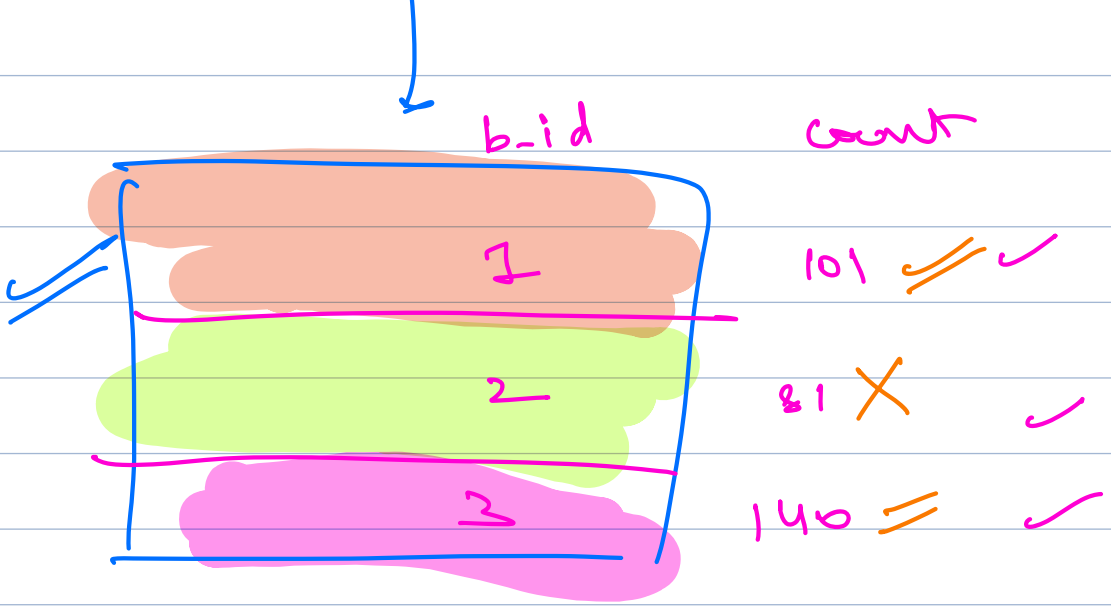
Q Print the batches names along with count of student that have more than 100 students.






```
select count(s.id), b.name
from student s
join batches b
on s.b_id = b.id
group by b.name
```





where → used to filter rows
Not GROUPS

HAVING → allows you to do filtering on groups.

```
select  b.name
from student s
join  batches b
 on  s.b_id = b.id
 group by b.name
 having  count(s.id) > 100;
```

A = [ ]

B = [ ]

ans1 = [ ]

```
for row1 in A :
    for row2 in B:
        if cond1 is true b/w A & B:
            ans1.add( row1 + row2)
```

ans2 = [ ]

```
for row3 in ans1 :
    if row3 satisfies cond2 :
        ans2.add (row3);
```

→ Map < Group by colm) : Double > avg Map

for each row in ans2 : } fill the map

```
for each group:
    if Cond. is true:
        print( avgMap( group)).
```

FROM

↓

JOIN

↓

WHERE

↓

Group by

↓

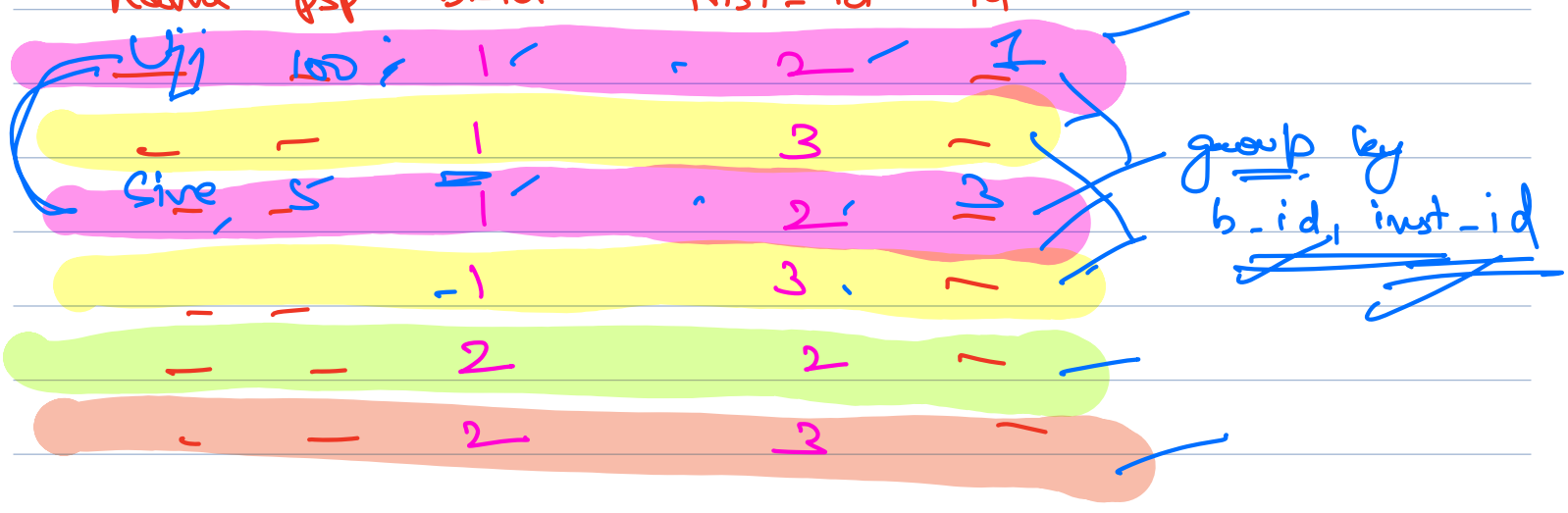
having

↓

Select (order by)

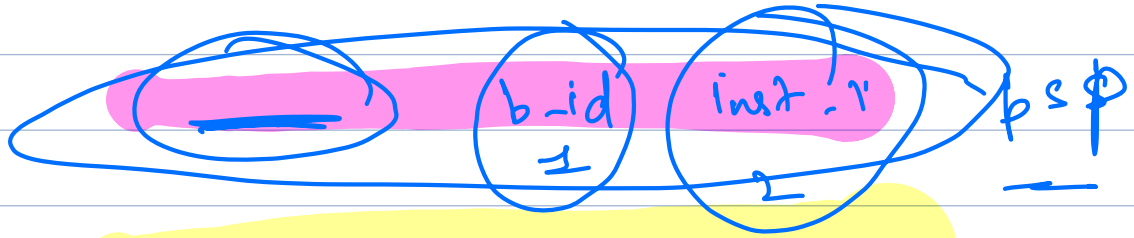
order of execution

* we cannot use where after group by as we only have groups/ set to no group.



select

group by


b_id
1
inst_v
2