Aggregations & Built-in functions

Agenda.

Aggregate fns
- group by
- having
- having vs where

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

Built-in fns - (Maybe)

⇒ Till now whatever SQL query we were writing were to get few of the rows in a table/tables.

Q find the students who...
find the batches ...—

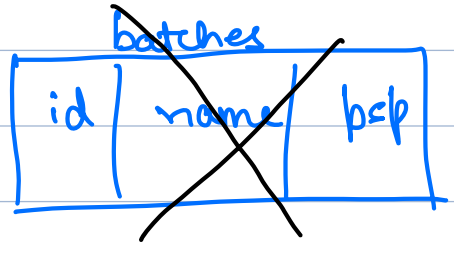
Do we always have to find exact entities?

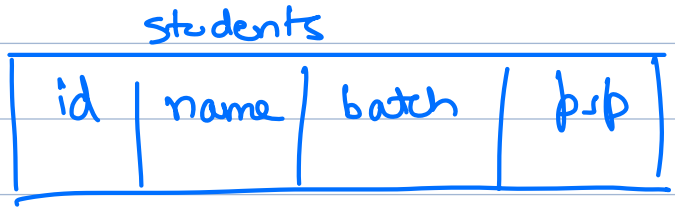
Sometimes, instead of getting a few of the entities, we have to get some analysis out of some entries.

↓

get data that is formed as a combination of rows.

eg. Get avg psp of every batch of scaler.





↓

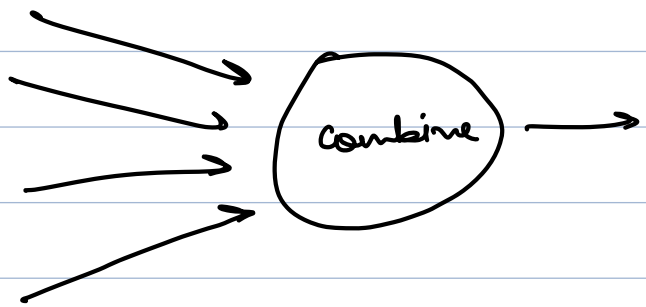
How would you have found that in code?

go through students table
↓
combine rows of students of a batch
↓
Calculate their avg

Aggregate fns.

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

count(_ _ _ _) ⇒ ①



① Count

students

id	name	age	batch_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 → takes a lot of values and combines them into a single value which is equal to count of non-null values in the set.

count (1, 2, 3, 4, 5) ⇒ 5

How to use count ?

↓

we use select to PRINT anything in SQL.

Query.

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

Similar to for loop on table students

Students

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

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

count(1, 1, null, 2) ⇒ 3

⇒ All aggregate functions ignore non-null values.

Q select count of students where age < 23.

```
select count(batch_id)
from student
where age < 23;
```

This will get all the values that are provided by from.

Students

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

count(1, 1, null) → 2

→ Aggregation happens at the end.

↳ first (ons) table is created and 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

select count of students with batch name of A.

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

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

Select ____
from A
join B
on cond 1
where cond 2

① ②

```
A = [ ]
B = [ ]
ans1 = [ ]
for row1 in A:
    for row2 in B:
        if cond1 is true b/w A and B:
            ans.add (row1 + row2)
```

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

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

} aggregate

print (count_s_id) → select

★ Remember on clause with where functions faster than only where clause because without on clause it functions like cross join which is inefficient in terms of space as well as time.

Other aggregate fns

→ you can print multiple aggregations at the same time.

```
select count(batch_id), sum(proxy),
       avg(psp), max(psp), min(psp)
from students;
```

⇐ whatever is the table.

from (x)

↘ whatever table is formed here

[concept helpful for subqueries class]

① MAX (. . . . --) ⇒ maximum value

② MIN (_ _ _) ⇒ minimum value

↑ should be comparable. eg int, date etc.

③ AVG ( )

Avg (1, 2, 3, null) → 2

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

6/4 = 6/3

Students	
id	psp
1	1
2	2
3	3
4	null

Q How many students are there?

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

constraint → no single coln that will always be not - null.

* → represents every column

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

select count(1) ⟹ prints count of all the rows
from students ;

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

checks if s_id is not null, will always be true for 1 not null

faster as no comparison

select count(1) == select count(*)

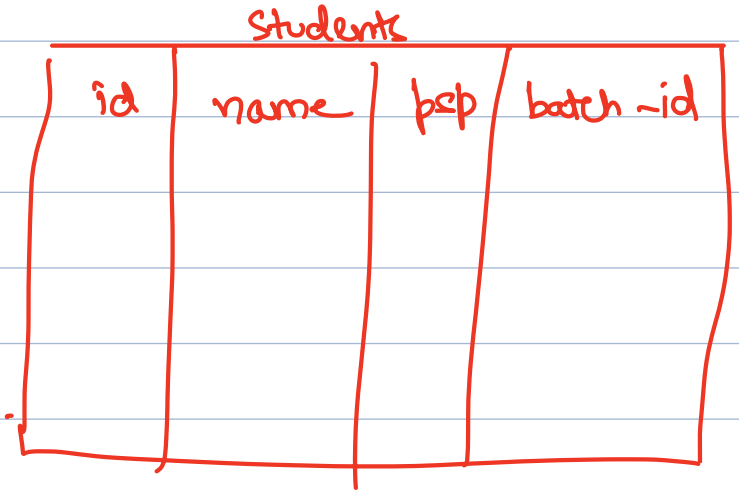
1 can never be null

* can never be null

count(1) == count(2) ✓

count(70) count(true)

— Till now, in aggregate fns we had a lot of values, that we aggregated / combined to a single value.



Q Get the count of students

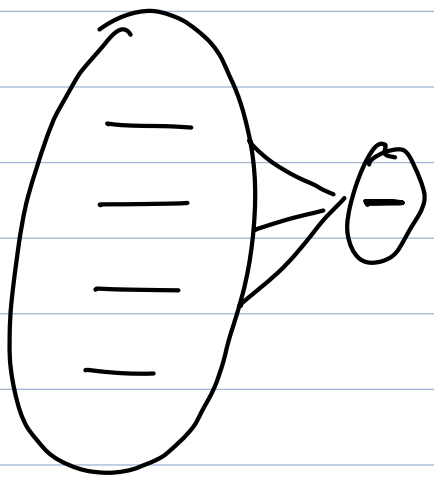
→ select count (*)
from students ;

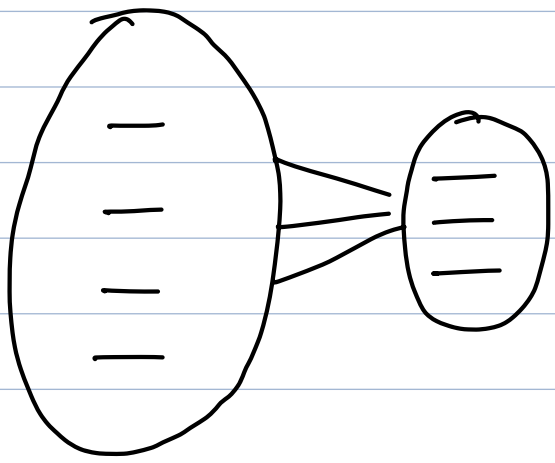
Q Get the count of students for every batch.

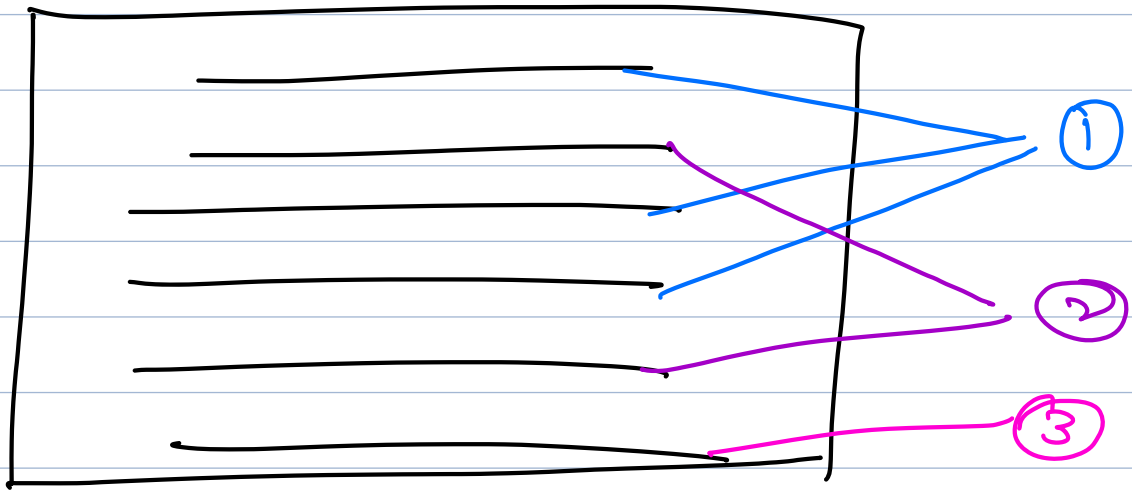
batch_id	count
1	50
2	10
3	60
4	40

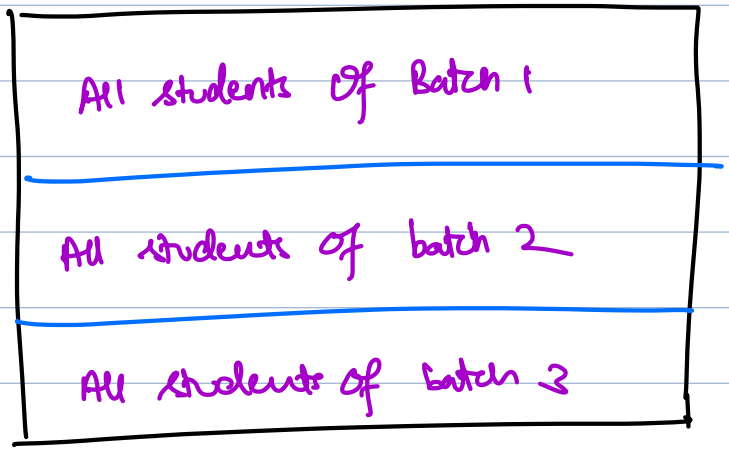
← something like this

Each of these sets will be independent → one row can't contribute to 2 sets.









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

Eg
```
group by batch_id;
// bring all of the rows with same batch_id together

group by batch_id, instructor;
```


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

join | where whatever

include cols in group by

eg.

```
select count(*), batch_id
from students s
join batches b
on ____
where
group by
```

students

id	name	b_id	psp
1	A	1	30
2	B	2	40
3	C	1	30
4	D	1	50

b_id	count
1	3
2	1

⇒ incomplete result

Can't understand the result

⭐ ⟹ You can only use those cols that are present in group by.

↓

All values of the same group have only one / those value in common that were in group by statement.

[CODE]

→ films released in each year
→ films per year per rating
→ ↳ same ques + rental > 4

```
A = [ ]
B = [ ]
ans1 = [ ]
for row1 in A:
    for row2 in B:
        if cond1 is true b/w A and B:
            ans.add(row1 + row2)

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

this is where grouping happens

count_s_id = 0

```
for row in ans2:
```

aggregate

if row[s_id] not null:
count_s_id += 1

print (count_s_id) → select

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

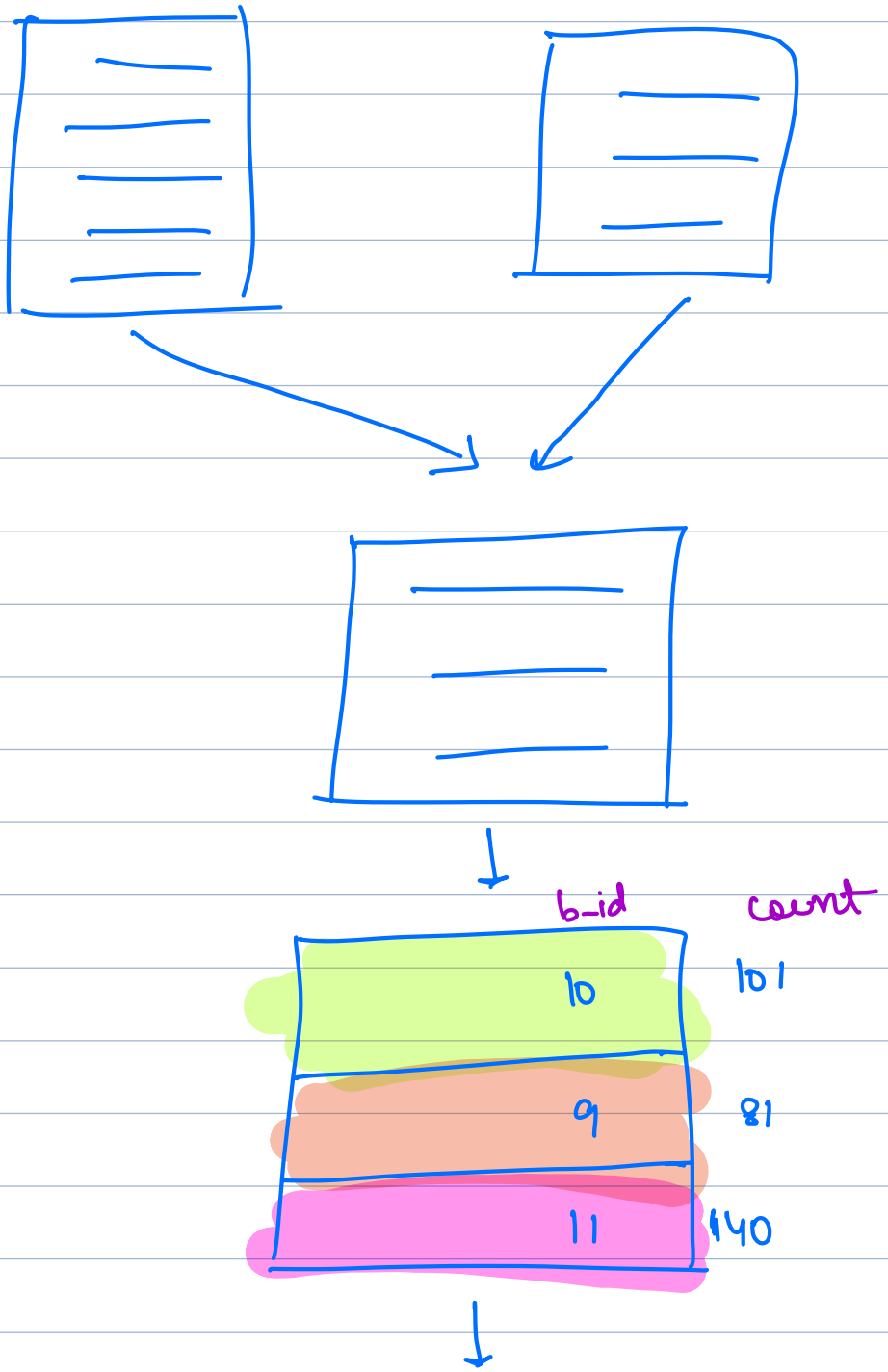
students

id	name	age	b_id

batches

id	name

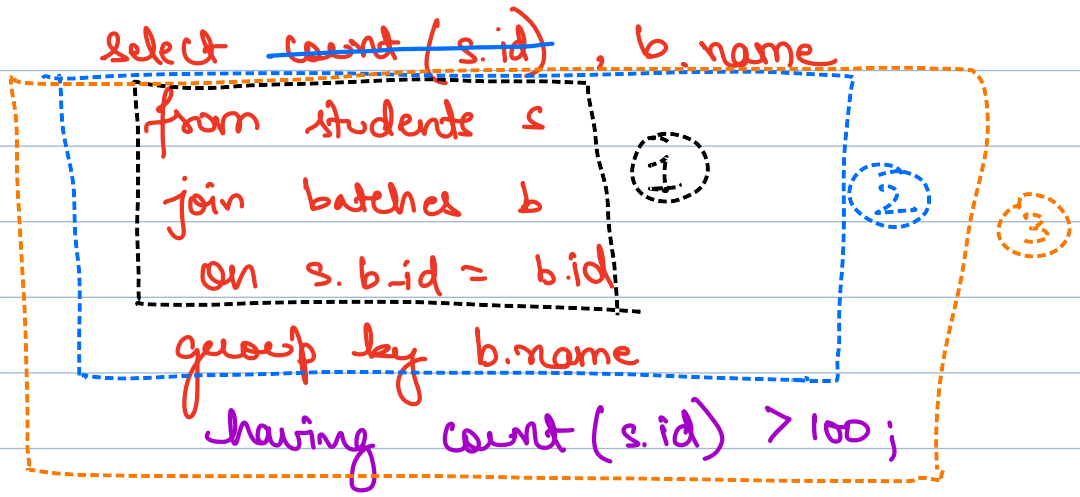
```
select count(s.id), b.name
from students s
join batches b
on s.b_id = b.id
group by b.name
```



have I been asked to print all of them? NO

where : used to filter rows not groups

HAVING → allows you to do filtering on groups.




```
A = [ ]
B = [ ]
ans1 = [ ]
for row1 in A:
    for row2 in B:
        if cond1 is true b/w A and B:
            ans.add(row1 + row2)

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

```
Map<(Group by cols): Double> avgMap
Map<(GroupBy cols): Int> maxMap
```

for each row in arr 2:] // fill the maps

for each group :
    if "cond" is true : → HAVING.
        print (avg Map (group))

FROM
↓
WHERE
↓
GROUP BY
↓
HAVING
↓
SELECT

order of execution.

[CODE]

→ count of movies per year per rating & rental > 4
also count > 130.

→ use avg with having
↓
avg (rental) > 4.

used group by without aggregations → to print all the possible groups [set].

Q can we use where after group by ?

NO → as rows are gone and we only have groups.

[Please solve your assignments]