Like

Whenever we want to compare strings, we can use 'Like' query

Ex: Batches

id	BatchName
1	AugBeginner22
2	AugInter22
3	AugAdv22
4	MayBeginner22
5	USAugBeginner22

* get all batch ids that started in August

(%) → percent
(_) → underscore

Select * from batches where batch_name like '%Aug%';

% ⇒ any no. of any character

% text % ⇒ text is not case-sensitive

_ ⇒ (1) any single character

%Aug% → %(≥0) Aug %(≥0)

XAugY ✓
AugXY ✓
XYAug ✓
Aug ✓

Aug

XAugY → ✓
XYAug → ✗
AugXY → ✗
Aug → ✗

⇒ LIMIT

⇒ Photo ⇒ Instagram (Virat)

↓

10k comments

↓

Comments

id	Comment	post_id	Likes	description

post_id = 10

```
Select * from comments where post_id = 10
Order by likes desc limit 5;
```

↓

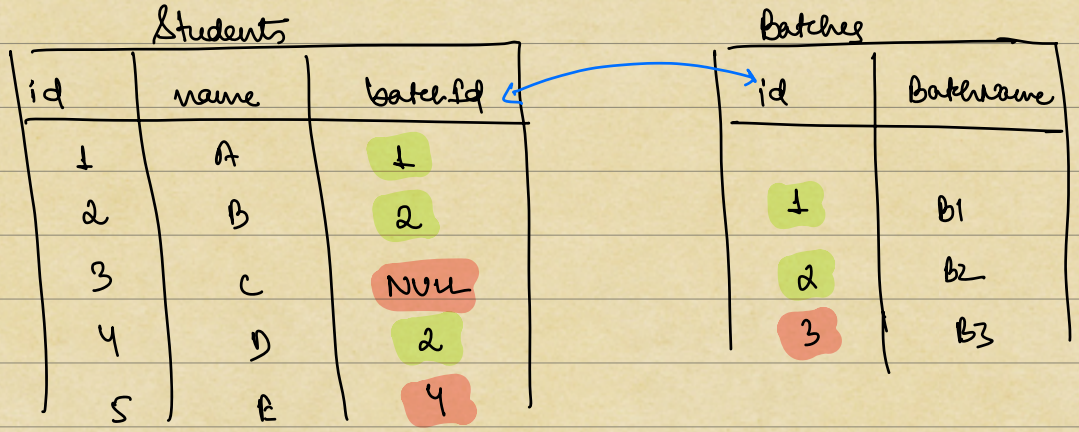
This will fetch all comments for given post_id and order it by no. of likes, then return the top 5 results

⇒ OUTER JOINS:-

- Left join
- Right join
- Full join

* Left join:-

If a row on the left side, doesn't match with any row on the right side, it will still be added with null values



O/p.

id	name	batchId	id	BatchName
1	A	1	1	B1
2	B	2	2	B2
3	C	NULL	NULL	NULL
4	D	2	2	B2
5	E	4	NULL	NULL

* Right join:-

If a row on the right side, doesn't match with any row on the left side, it will still be added with null values

Students

id	name	batchId
1	A	1
2	B	2
3	C	NULL
4	D	2
5	E	4

Batches

id	BatchName
1	B1
2	B2
3	B3

O/p.

id	name	batchId	id	BatchName
1	A	1	1	B1
2	B	2	2	B2
4	D	2	2	B2
NULL	NULL	NULL	3	B3

* Full join

Adds all the columns from both left and right side, even if the conditions do not match.

Students

id	name	batchId
1	A	1
2	B	2
3	C	NULL
4	D	2
5	E	4

Batches

id	BatchName
1	B1
2	B2
3	B3

O/p.

id	name	batchId	id	BatchName
1	A	1	1	B1
2	B	2	2	B2
3	C	NULL	NULL	NULL
4	D	2	2	B2
5	E	4	NULL	NULL
NULL	NULL	NULL	3	B3

* left =>

```
Select  { columnNames }
from    { table Name } t1
left join { table Name } t2
on  t1.column = t2.column;
```

* right =>

```
Select  { columnNames }
from    { table Name } t1
right join { table Name } t2
on  t1.column = t2.column;
```

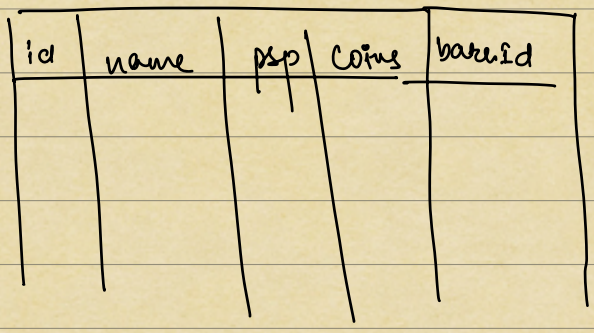
* full =>

```
Select  { columnNames }
from    { table Name } t1
full outer join / full join { table Name } t2
on  t1.column = t2.column;
```

* Natural join, cross join, Update table, Alter table, date

⇒ AGGREGATION :

grouping, collecting, combining



Q1. Find the Student with max psp.

Q2. Find the avg psp of the Students for all batches

Q3. Find the total no. of Scaler coins by Students for all batches

Aggregation Functions :

fun (set of values) ⇒ single value

sum (1, 3, 5, 7) ⇒ 16

max (1, 3, 5, 7) ⇒ 7

min (———) ⇒ 1

avg (———) ⇒ 4

count (———) ⇒ 4 ⇒ no. of elements

* aggregate func. only work on non-null values

eg ⇒ count(1, 6, NULL, 7) ⇒ 3 ✗

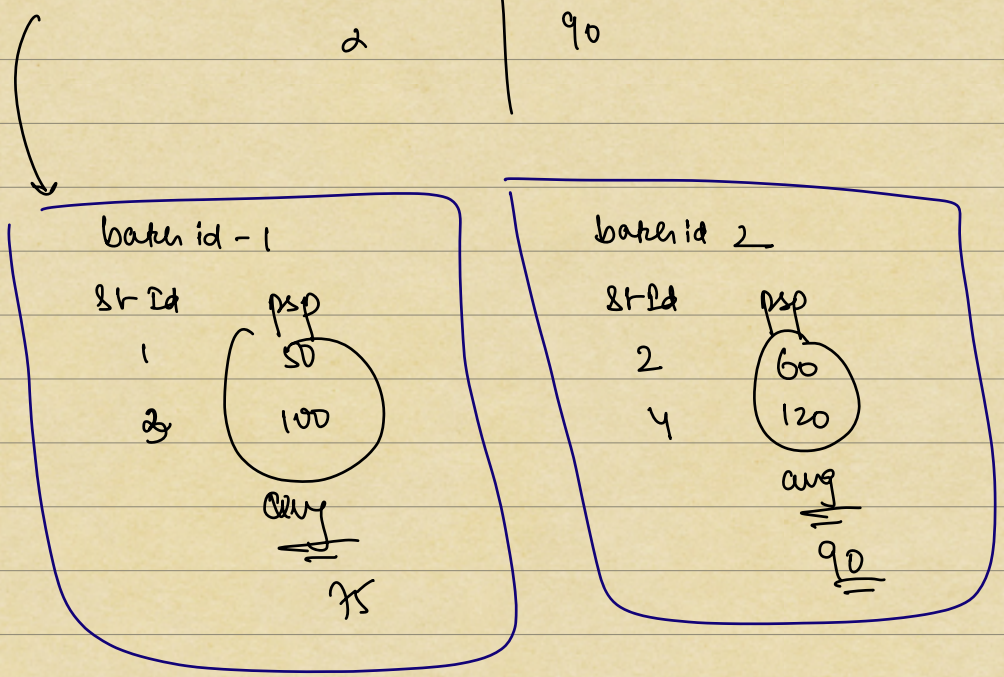
Q.4. Find the total no. of Students in a particular batch

Till now, we were running our aggregate functions on the entire table

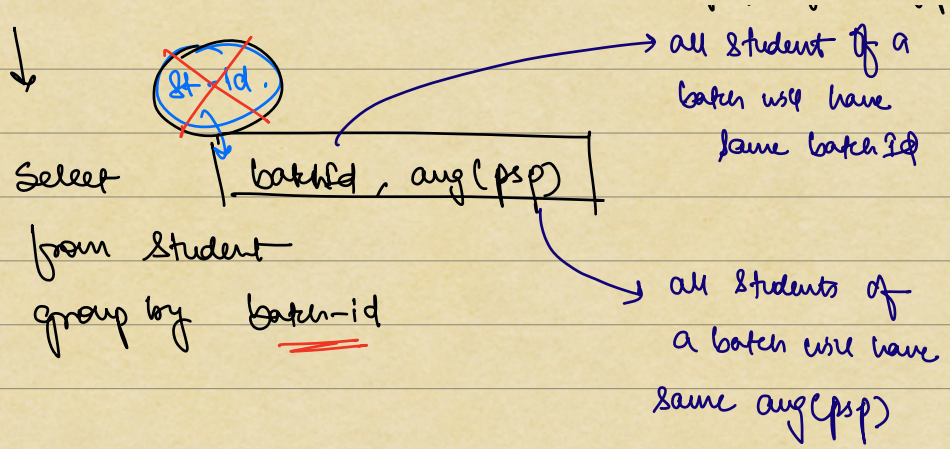
id	name	batch id	psp
1	A	1	50
2	B	2	60
3	C	1	100
4	D	2	120

O/p

batch Id	avg psp
1	75
2	90



* We need to create groups of Students by batch Ids, and then run avg( ) ← func on psp of each grp.



Batch-1		avg
A	50	70
B	60	70
C	70	70
D	80	70
E	90	70

Q.4. Find the total no. of Students in a particular batch

count(*) → any column is not null in row, count that

Select batchId, count(emailId)
from Student
group by batchId

if email of a student is null, that student must be counted

* find the avg batch psp per batch

* find the batches whose avg psp ≥ 80.

Option 1

```
Select batch_id, avg(psp)
from student
group by batch_id
where avg(psp) >= 80
```

Execute ⇒ X

where clause runs on actual table data, there is no column called avg(psp)

Option 2

```
select batch_id, avg(psp)
from student
where psp >= 80
group by batch_id
```

Execute ⇒ ✓

Accurate ⇒ X

Neither are correct

* where clause cant be used after group by

group by batch id

1, A B C

2. D E

id	name	batch Id
1	A	1
2	B	1
3	C	1
4	D	2
5	E	2

⇒ HAVING

If we have to filter grouped data, we use HAVING

* find the batches whose avg psp ≥ 80.

```
Select batch-id, avg(psp)
from Students
group by batch-id
having avg(psp) ≥ 80
```