JOINS I

Agenda.

- Update
- Delete
  - Drop
  - Delete
  - Truncate

- Joins
  - INNER JOINS
  - Self Joins
  - Outer Joins
    - Right Join
    - Left Join
    - Full Join
  - Cross Joins
  - Syntactic Sugars
    - USING
    - NATURAL join
    - IMPLICIT JOIN
  - Joins with where

UPDATE

→ Change data that is already present in the table in the database

'UPDATE' clause

update {table_name}
set {new-value of a particular coln or a set of columns}
where {where_clase}

student

id	name	psp

```
update students
set psp = psp + 10
where id=1
```

⇒ if you miss the where clause, it will update the coln in each of the rows.

⇒ Update a set of columns

```
update students
set name = 'Ujjwal', psp = 80
where id = 1;
```

⇒ Update multiple rows at once

```
Update students
set name = 'Ujjwal', psp = 80
 where id<100;
```

[CODE]

→ update multiple columns at once

→ update all rows at once

DELETE

→ it always deletes a complete row.
→ it will not delete a table
→ instead used to delete rows in a table

```
delete from {table_name}
    where {condition};
```

⇒ what if we forget the where clause, all the rows will be deleted.

eg.
```
delete from film
    where title = 'Deepak';
```

[CODE]

→ foreign key issue
→ delete all rows

Code of delete :

```
for each row in film:
    if row matches cond in where:
        delete row;
```

Delete vs Truncate vs Drop

Delete ⇒ deletes few of the rows in a table

↳ rows that match a given condition

TRUNCATE ⇒ truncate {name of the table}

⇒ all of the rows will get deleted

→ similar in out come to delete from {table name}

⇒ faster than delete to delete all rows.

Delete ⇒ deletes rows one by one

→ possible to undo an operation.

→ does not change auto increment sequence.

truncate ⇒ it is faster because it doesn't go row by row. Instead it vanishes the prev table and create a new table. Instead it vanishes the prev table and creates a new table.

→ Not possible to undo an operation.

→ resets auto increment sequence

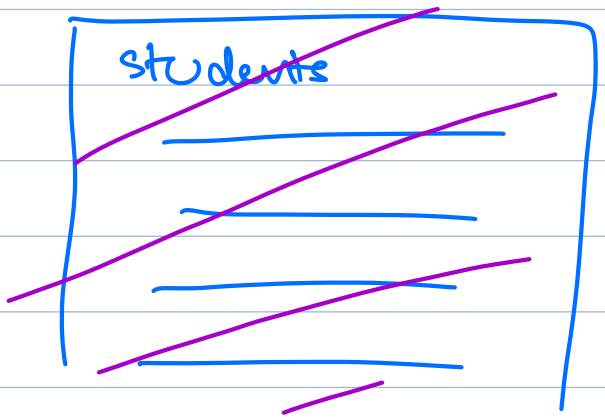
arr

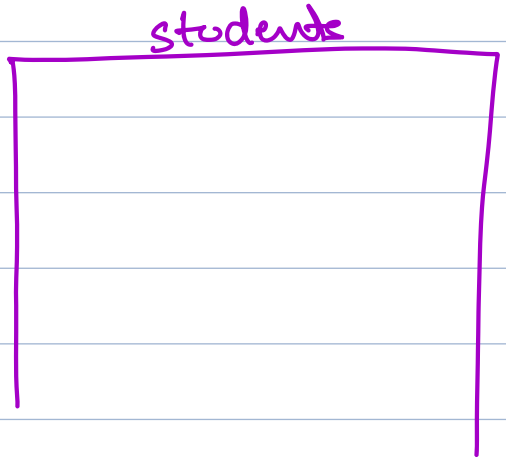
⇒ 

⇒ return an empty array

option 1 → delete elements one by one and return arr

option 2 → return a new array





SQL has this log file that has track of all activities - deletes, update whatever, this helps us to undo an activity.

★ we will learn rollback in transaction classes.

Students

id	name	email
1	a	x
2	b	y
3	c	z

4 d a

→ delete from students

→ insert into students (
⇒
)

1 d a

VS

truncate students
insert into students (
)

Delete from	Truncate
① Deletes row by row	completely removes the older table and creates a new table with clean look
② Slow	② faster
③ Doesn't reset the key	③ Resets the key
④ undo a row that was deleted	④ cannot do an undo

(5) can delete specific row | (6) always deletes the entire set of rows.

DROP

→ doesn't just delete the data
→ deletes the table.

* After delete / truncate you are still left with a table but with no data, with drop you loose all the data as well as the table.

Example.

A → B relationship

↓

have some problems

Team delete

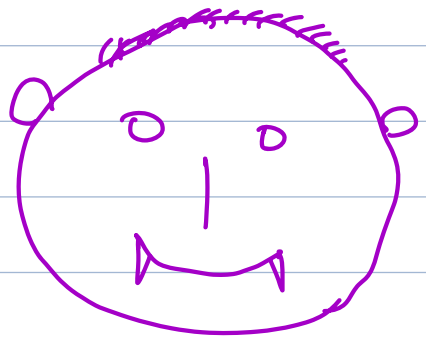
→ go through each of the problems in a rel^{n} 1 by 1 and try to resolve them

Team truncate → breakup

→ new relationship A → C
B → D

Team drop → break up
→ Hogy se single

DONE WITH CRUD



JOINS

→ till now queries on single table

Students

id	name	b_id
1	John	1
2	Jamie	1
3	Jim	2
4	Jenny	3
5	Jack	2

batches

id	name
1	A
2	B
3	C

Q Print name of every student with name of their batch.

John	A
Jane	A
Jim	B
Jenny	C
Jack	B

students batches

Joins → Allow to combine data across multiple tables

Students

id	name	b_id
1	John	1
2	Jane	1
3	Jim	2
4	Jenny	3
5	Jack	2

batches

id	name
1	A
2	B
3	C

1 John 1 1 A
1 John 1 2 B
1 John 1 3 C

⇒ I want to be able to stitch data of a row of students and data of a row of batches together if students' b_id = batch's id.

⇓

Join condition.

```
select  students.name, batches.name      <- fill this at the end
  from students
   join batches
   on students.b_id = batches.id ;
```

ans = [{---,---}, {---, ---}]

```
for each row1 in students:
    for each row2 in batches:
        if cond^n is true:
            ans.add (stitched row)
```

~~print (ans)~~

```
for each row in ans:
    print ( row [stu.name] , row [b.name])
```

Students

id	name	b_id
1	John	1
2	Jane	1
3	Jim	2
4	Jenny	3
5	Jack	2

x x x ✓ ✓

✓ x ✓ x x

x ✓ x x x

batches

id	name
1	A
2	B
3	C

```
from students
 join batches
  on students.b_id = batches.id
```

ans. (internal table)

student			batches	
id	name	b_id	id	name
1	John	1	1	A
2	Jane	1	1	A
3	Jim	2	2	B
4	Jenny	3	3	C
5	Jack	2	2	B

$O(n * m)$ ⇒ worst case

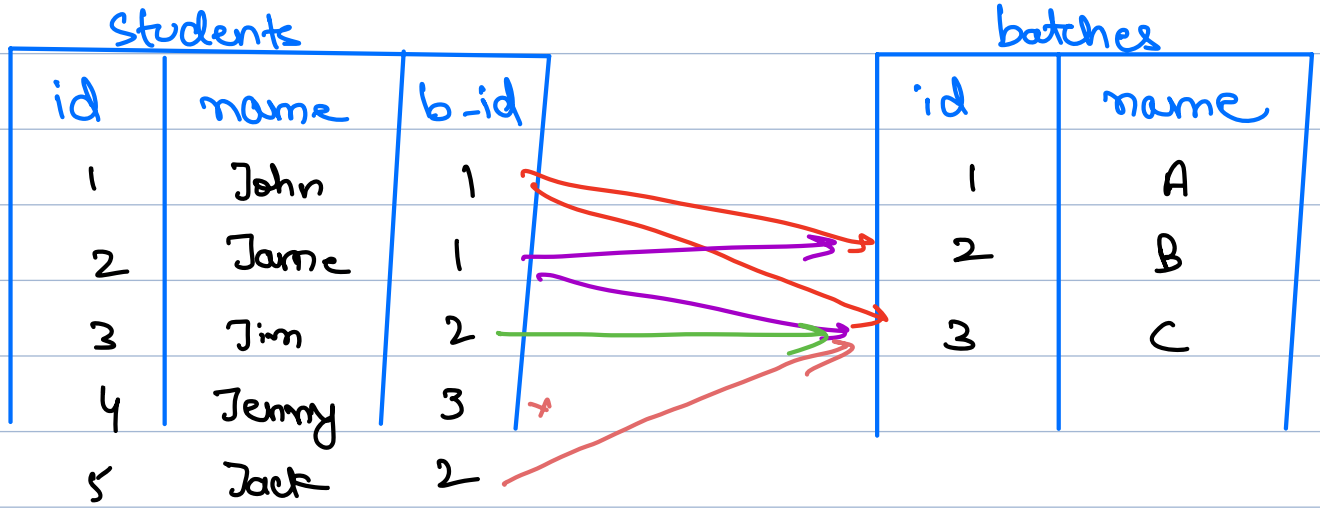
```
select * from students
    join batches
on students.b_id = batches.id
```

↳ Prints the entire above table

```
select
from students
    join batches
on students.batch_id < batches.id;
```

Q How many rows will it give for above table?

6



```
select
from students
join batches
on length(student.name) > length(batches.name)
```

* Joins have nothing to do with PK and FK.

[CODE] → join with film & language - id

ALIAS

→ Can name a table something to make query look simpler ;

[CODE]

Student Buddy

→ we assign a senior student as a buddy to you

student

id	name	batch_id	psp	buddy_id
1	naman	1		3
2	Amit	1		3
3	Ajeet	2		null

→ A buddy is nothing but a student
→ A buddy_id is nothing but student_id.

Buddy_id is a self referential foreign key.

Q for every student print their name along with their buddy's name

Naman	Ajeet
Amit	Ajeet

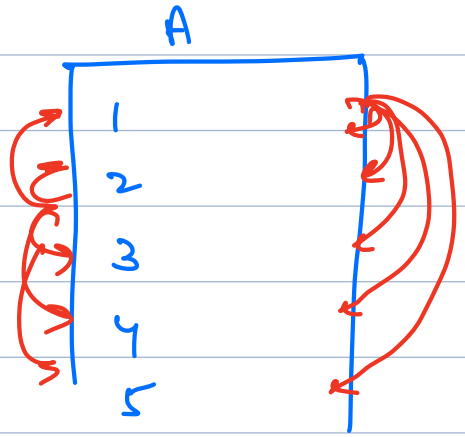
buddies

id	name	batch_id	psp

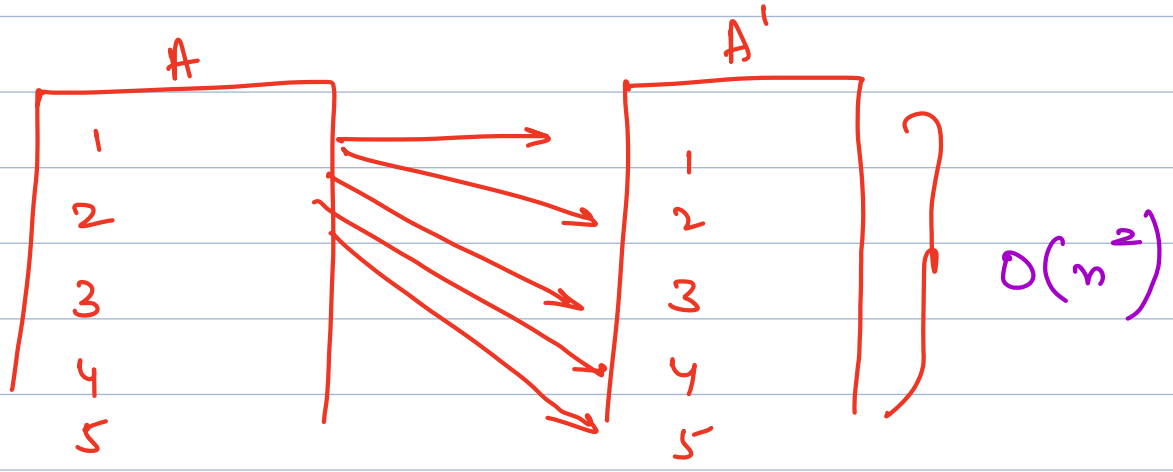
if buddies is a separate table

```
select s.name , b.name
from student s
 join buddies b
on s.buddy_id = b.id
```

Self Join ⇒ Combine rows of a table with itself.



from A
join A




```
select  s.name , b.name
from    student s
 join   student b
 on     s.buddy_id  = b.id
```

If we do a join, can we avoid alias?

NO

alias are mandatory in self join.

Another example

↳ employee table. → manager

employee

id	name	email	phone	mgr_id