→ ANY with Subquery :-

Select columnName(s) from TableName &
WHERE columnName [Operator] ANY
(select columnName from TableName
WHERE Cond^n);

Now here while using any :- this operator is very important to be mentioned :- like what type of comparision you want to have?
So this operator must be a standard comparision operator → (=, <>, !=, >, <, >=, <=)

So the query should look like :-

columnName > any (———)
— " — >= any (———)
— " — <= any (———)
— " — != any (———).

Ex: Consider the OrderDetails table
Orderdetails (orderDetailID, OrderID, productID, Quantity)
and also the products table (ProductID, ProductName, SupplierID, CategoryID, Unit, price)

So: find the ProductName of all those products which have their orders qty larger than 5/0

So the query :-

```
select ProductName from products where
    ProductID IN ( select productID from
    order details where Quantity > 20)
```

= ANY

→ apart from IN operator we can also use "= ANY"

Ex: Find the ProductName of all those products which have their productIDs less than ~~than~~ any of the product having orders quantity equal to 1.

query :-

```
Select ProductName from products where
ProductID < any ( select productID
    from orderdetails where quantity = 1)
```

Now this is how "any" keyword works!

→ Exactly same thing works for all other relational operators too!

and ANY = works like OR } less than "any" (↓)

"< any" → any of these multiple values.

Now ALL :- works like AND | < ALL

~~AND~~ → less than the all the values we got from subquery.

→ ALL with Subquery :-

find the Product Name of all those products which have their ~~order quantity larger than 50~~ product IDs less than all of the product having orders Quantity equal = 1.

Soln.

```
Select ProductName from products
where ProductID < ALL
(select productID from orderdetails
 where quantity = 1)
```

Operator	Meaning
= ALL	→ Means equal to each & every value of the subquery.
	↳ Now this is highly improbable.
	↳ our subquery will return multiple distinct values then No any value from outer query will be equal to all the value of the subquery.
> ALL	→ means greater than 1st, 2nd, 3rd ... nth value we got from the subquery.
	↳ greater than each & every value.
	↳ greater than maxm of Inner query.
< ALL	→ less than minm of Inner query.
	↳ Subset data of Inner query.

>= ALL → greater than or equal to max^m of Inner query.

<= ALL → less than or equal to min^m of Inner query.

Operator	Meaning
= ANY	→ IN (Inner query)
< ANY	→ < → lesser than the max^m of the Inner query.
> ANY	→ greater than the min^m value of ~~the~~ Inner query.
>= ANY	→ greater than or equal to the min^m value of Inner query
<= ANY	→ lesser than or equal to the max^m value of Inner query

---

→ These ALL and ANY are not much used practically so instead of them :- max^m and min are used only in the inner query and same → max & min are used just before the brackets of outer query.

Qu:- consider employee (employeeID, FName, LName, JobID, sal., DeptID).

Find all the employees whose salaries are greater than the salary of all employees in the sales dept with deptID = 2.

Soln.

```
Select * from Employee where
sal > all ( Select sal from Employee
            where deptID = 2 )
```

Now another way of writing this :-

```
Select * from Employee
where salary >
( Select max(sal) from Employee where
            deptID = 2)
```

→ Exists with Subquery :- Used to test for the existence of any record in a subquery.

↳ This will return true if the subquery has or returns one or more records.
otherwise this will return false.

Syntax :-

```
Select columnName(s) from tableName
where EXISTS ( Select col.Name
from tableName where cond^n )
```

If this query has some records than true

and if Exists returns true :- then the outer query will be performed

↳ its just like the "If - else" stmt.

if exist = true → then outer query will be performed !

Ex:-

Select * from customers Exists → true

when exists returns true ↓ this will be performed

(select * from customers where customer ID = 1)

↳ this gives us 1 record

↳ so exists will return = true

⇓ equivalent to

Select * from customers where TRUE.

Condn = true here

Hence fetch all customers records

and if exist = false :- then No any records will be fetched from the outer query.

and when Using NOT with exists :-

false will change to true.

Ex:- Select * from orders where

(Select NULL)

↳ Now this will give us :-

Result : NULL → column
NULL → record } so one record is fetched

Hence exist will give us ↳ TRUE

Ex: Consider customers and orders table.
write a query to select all such customers record which have atleast one order placed.

Soln. Here we don't have any column which can count the orders of the customers

→ order placed = not any column.

So the query :-

```
Select * from customers
where customerID IN
(select distinct customerID
from orders)
```

Now this exact same query can be written using exists also

```
Select * from customers where exists
(select * from orders where customers.CustomersID
= orders.CustomerID)
```

↳ Inner query runs for each record of outer query

~~No previous~~

Set operators :- Used to combine results from two or more Select statements

→ So the operators are :-

1. Union 2. Union all 3. Intersect

4. ~~minus or~~ minus or except.

Consider :-

Table 1

C1	C2
1	1
2	3
4	5
3	6

Table 2

C1	C2
1	4
2	3
4	5
4	

Now

```
(Select * from Table 1)
Union
(Select * from Table 2)
```

→ Set Operators are nothing But :- vertical merging.

So here :- doing union :-

will merge the table and common records will be written once only and all remaining records will be fetched

Resultant

C1	C2
1	1
2	3
4	5
3	6
2	4
4	3

***

Now one very imp. point :- all select statements must have equal No. of columns.

→ Now Union all :-

```
(Select * from table 1)
   Union ALL
(select * from table 2)
```

} Here all the records will come and also the duplicate records too.

C_1	C_2
1	1
2	3
4	5
3	6
1	1
2	4
4	3
4	5

→ Intersect : Only the common records will come :-

↳ complete row must be common / same.

```
(Select * from table 1) Intersect (select * from
                                   table 2)
```

Result set

C_1	C_2
1	1
4	5

→ minus or except :-

(select * from table 1) minus (select * from table 2)

↳ fetching all the records of table 1 and ignoring/avoiding the common records of Both tables!

C_1	C_2
2	3
3	6

* In some SQL servers :- "minus" will work and in some other servers :- "except" will work.

18/08/2022. # Lesson 12 #

Que. R(A, B, C, D, E)

Key ⇒ AC, AD,

So some seper keys of R = { ACE, ACD, ABC, ABD, ADE, ABCD }

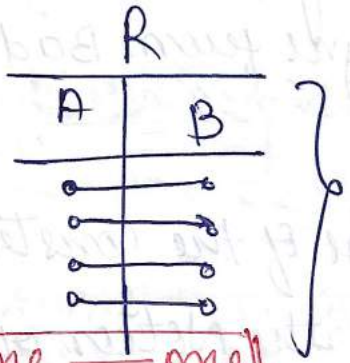
Que. Employee [EmpNo. , EMPName, dept. , Sal.)

Primary Key: EmpNo. → then through EmpNo. we can easily identify one record uniquely.

functional dependency :-

consider a relation R & 2 attributes in Relation R = A & B.

Now, B is functionally dependent on A (denoted By A → B), if each value of A is associated with exactly one value in B in relation R.



A → B.

one — one association must be there for two attributes :- for functional dependency of one attribute to another.

→ for ex :- in previous question's employee table :-

Emp No. → EmpName ✓

Holds good

But

EmpName → Emp No ✗

This will not hold right ✗

Ex:- Consider the table:

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_2	c_2	d_3
a_3	b_3	c_2	d_4

Consider then following functional dependies:-

1. ✗ $A \to B$ ✗ doesn't Holds
2. ~~✓~~ $A \to C$ → Holds good.
   - $a_1 \to c_1$
   - $a_2 \to c_2$
   - $a_3 \to c_2$ } one-one
3. ~~✗~~ $B \to C$
   - $B_1 \to C_1$
   - $B_2 \to C_1$
   - $B_2 \to C_2$ ✗
4. ~~✗~~ $C \to D$
5. ✗ $C \to A$
5. ✗ $AC \to D$ → $a_1c_1 = d_1$ & $a_1c_1 = d_2$ also ✗
6. ~~✓~~ $AB \to C$ ✓ → Holds good.

→ Some important points:

1. functional dependencies play a key role in differentiating good DB designs from Bad DB designs
2. A functional dependency is the type of the constraint that is a generalization of the Notion of Key.
3. $X \to Y$, where X is a set of attributes that can determine the value of Y.

→ means through the FD constraint we are able to derive the Candidate Key!
& through Candidate key :- all other Super keys!

Closure of an attribute 'A' :- it is the set of attributes which we can drive from A.

Ex: consider the relation R(A, B, C, D, E)

FD : { A → B, C → E, C → D, B → E }

Now what is the closure of attribute B :-

B^+ = denoted

So $B^+ = \{B, E\}$

→ closure of $C^+ = C^+ = \{C, D, E\}$

→ closure of $A = A^+ = \{A, B, E\}$

↳ from A :- we can drive B

& from B :- —"— E

Hence from A :- we can drive 'E' also.

→ Closure of an attribute set AC :-

$AC^+ = \{A, C, D, E, B\}$

→ trivial functional dependency :- Some functional dependencies are said to be trivial because they are satisfied by all relations.

Ex: A → A, B → B ⇒ So these are trivial functional dependecies.

Armstrong's Axioms :- RAT

R = reflexive
A = augmentation
T = transitivity

#01. Reflexivity Rule :-

A functional dependency $\alpha \rightarrow \beta$ will always holds if : $\beta \subseteq \alpha$ where α, β are attributes or set of attributes in a relation R.

Ex :- $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow AB$

all these are trivial FDs also.

#02. Augmentation Rule :-

A functional Dependency $\alpha \rightarrow \beta$ holds, then $\gamma\alpha \rightarrow \gamma\beta$ also holds where γ, α, β are attributes or set of attributes in a relation R

Ex : if $A \rightarrow B$ ⇒ then $AC \rightarrow BC$ also holds
if $AB \rightarrow C$ ⇒ — " — $ABD \rightarrow CD$ good. — " —

#03. Transitivity Rule :- A functional dependency $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ also holds, then :-

$\alpha \rightarrow \gamma$ also holds !

where α, β & γ = attributes or set of attributes

Ex :- $A \rightarrow B$, $B \rightarrow C$ } then $A \rightarrow C$ ✓

→ Additional Rule :-

Union Rule :- A functional dependencies $\boxed{\alpha \to \beta}$ and $\boxed{\alpha \to \gamma}$ holds, then $\boxed{\alpha \to \beta\gamma}$ also holds

↳ union only in RHS

Ex: If $\boxed{A \to BC}$ holds then :- $\boxed{A \to C}$ & $\boxed{A \to B}$ also holds. ✓

and

if $AB \to c$ holds then :- $\boxed{A \to C \ \& \ B \to C}$

↳ we cannot say that these FDs will hold or not!

---

closure of a set of functional dependencies :-

↳ Here you have find all such dependicies which you can derive from the given functional dependicies of the given table or relation.

Consider :- $R(A, B, C, D)$

its FDs = $\{A \to B, B \to C, AB \to D\}$ all

Now we have to derive the FDs from the given set of FDs :-

↳ functional dependency closure.

F^+ = FDs = $\{A \to C, A \to D\}$

→ pseudo transitivity :- if $A \to B$ and $AB \to D$ Holds then :-

$\boxed{A \to D}$ also holds

Ques. Given $R(A, B, C, D, E)$ & FDs = $\{A \to B, A \to C, CD \to E, B \to D, E \to A\}$ then which of the following FDs is not implied by above set?

Sol^n.

A. $CD \to AC$ &

$\Rightarrow CD \to E$ ✓ $E \to A$ ✓

$CD \to A$ ✓

$CD \to AC$ ✓ → because $C \subseteq CD$.

~~B.~~ $BD \to CD$

$BD \to D$ ✓ But $BD \to C$ ✗ } Not derived from given set.

C. $BC \to CD \Rightarrow B \to D \to BC \to CD$ ✓

augmentation rule.

D. $AC \to BC$.

similarly we also!.

$A \to B$ ✓

then $AC \to BC$ ✓

we can also do this question with the help of :-

closure of an attribute set :-

$AC^+ = \{A, C, B, D, E\}$

$BC^+ = \{B, C, D, E, A\}$

$CD^+ = \{C, D, E, A, B\}$

$BD^+ = \{B, D\}$

clearly $BD \to CD$ ✗ false.

19/08/2022 # Lesson 13 #

→ Composite Key :- means combination of two or more attributes.

Now consider the relation R(A,B,C,D,E)
one composite key = AB is our primary key

then :- 1. all values in A must be NotNull
2. — V — B — V —
3. — V — AB ⇒ must be Unique

→ you can use the subqueries with the following keywords :- 1. WHERE
2. FROM
3. SELECT also.

Select (_____) from _____
↓
subquery.

→ there are two ways of writing the subquery :-

Interrelated
→ if somehow there is a connection b/w the sub query and the outer query
↓
for each record of outer Query, inner query runs everytime.

individual : When No any relation with outer query.
↓
So Subquery runs first then outer query uses its result
↳ most of the time we did this only!

Ex. of Interrelated :-

Select products.productName from products where exist (Select * from order details where products.productID = order details.productID).

In inner query we have used the outer query's table for comparision purpose.

↳ Hence for each value of this column :- the inner query runs.

So here we are matching the Outerquery's table & one column with the innerquery's table & its one column

↳ then this is K/as interrelated kind of query

Ques: Consider the Relation R(A, B, C, X, Y, F, Z, G)
given FDs = { A → B, A → Y, BC → XY, AYZ → G, Y → C }

Soln So we have :- A → BY
BC → XY
AYZ → G
Y → C

then FD^+ = { A → C, AZ → G, A → CBY, A → XY, A → X, }

finding keys Using FDs :-

consider the following table :-

R

A	B	C	D	E

$A \rightarrow B, C, D, E$

A = Key :- then A can drive :- B, C, D, E.

which means :-

$A^+ = \{A, B, C, D, E\}$

So what will do :- we will try to find out that which attribute or set of attributes have through which we can drive all other attributes of the table :- then that attribute or set of attributes is our key.

↳ firstly try to find out which attribute can drive all other column → one single attribute you try to find out.

Qu: consider R(A, B, C, D)

given FDs = $\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

then Find all keys.

Sol^n first we'll check $A^+ = \{A, B, C, D\}$ ✓

Hence A = our key

Now doing for B ⇒ $B^+ = \{B, C, D\}$ → cannot drive A

↳ so B = not our key.

for $C^+ = \{C, D\}$ → cannot drive A & B.

Hence Not our key.

So our key = A → Candidate keys.

Now all the super keys = A, AB, AC, AD, ABC, ABD, ACD, ABCD

↳ total = $2^3 \rightarrow 8$ → for 3 attributes (B, C, D)

Que. Consider $R(A, B, C, D)$
and given FDs = $\{A \to B, B \to C, C \to D, D \to A\}$

Now checking for A :-

$A^+ = \{A, B, C, D\}$ ✓ → yes A is our key

Now there is a FD :- $D \to A$ → means 'D' can derive A and we know that A derives everything

So D → also derives everything :-

$D^+ = \{D, A, B, C\}$ ✓ } D derives all

Hence D = our key also }

Now we can derive 'D' from C.

as $C \to D$ = given hence 'c' can also uniquely derives all other keys.

So 'c' also our key.

similarly B can derive 'C' & 'c' derives all.

So 'B also our key

finally : A, B, C & D → all are our Candidate Keys.

Now total Super Keys = $2^4 - 1$

↓ for 4 attributes (A, B, C, D)

1 → when No any key is selected as a super key

→ Null cannot be a key

→ So removing this possibility when No any attribute is involved in key.

Que. consider the relation R(A, B, C, D, E, F) and the candidate key = AB.

then the total No. of Super keys :-

~~AB/////////~~ Now with AB :-

each of the remaining 4 keys have 2-2 choices → included in a key or Not included.

Hence 2^4 choices :- 16

Que. consider R(A, B, C, D, E, F) and the FDs = {AB → C, B → D, C → E, CD → F}

Now in the given FDs :- find out the attributes which are not present in RHS of any FD.

⇒ A and B → present in LHS of 1st FD.

→ So No any key (apart from AB themselves) can derive AB → so only A & B can derive themselves.

So AB^+ = {A, B, C, D, E, F} → we can derive all the keys from AB

→ since A & B are not present in RHS :- then we cannot derive A alone and B alone from somewhere else → so combinely we taken AB as key

So the only candidate key possible here is = A, B

Que: consider $R(A, B, C, D, E, F)$ and $FDs = \{AB \to D, C \to B, D \to CF, B \to E\}$

Soln: Now here which attribute is not in right side :- **A**

and alone A = cannot derive something.

so we have to make combination with A.

→ making with B :-

$AB^+ \to \{A, B, D, E, C, F\}$

↳ Hence AB = our key.

Now checking for AC combination :-

$AC^+ = \{A, C, B, E, F, D\}$ ✓ Yes this also our key

Now $AD^+ = \{A, D, C, F, B, E\}$ ✓

So we have 3 candidate keys = AB, AC, AD.

Que. Consider Relation $R(A, B, C, D, E, F, G)$ and

$FDs = \{AB \to D, G \to A, D \to F, B \to E, E \to C, A \to G, C \to B\}$

Soln: Now toeying if A = alone is key or not?

$A^+ = \{A, G, \}$ = that's it = hence A alone not a key. ✓

Now toeying with B^+ of AB :-

$AB^+ = \{A, B, D, G, E, F, C\}$ ✓ Yes AB is our candidate key!

Now we can replace A by G :-

So GB or BG also our candidate key

Similarly :- AC, CG, AE and EG are also our keys.

So candidate Keys :- AB, BG, AC, CG, AE and EG.

Que. Consider a relation R(A, B, C, D, E, G) & FDs = {AD → E, AB → C, B → D, AC → B, EG → G, BC → A}

Soln alone A = Cannot Be the key

So trying AB ⇒

AB^+ = {A, B, D, E, C, G} ✓ Yes this is our key !

Now trying for AC :-

AC^+ = {A, C, B, D, E, G} ✓

& ofcourse I can replace A with B in AC.

then BC = also our key.

So our candidate keys = AB, AC & BC !

Que. Consider R(A, B, C, D, E, G) & FDs = {A → B, BC → D, E → C, D → A}

Soln alone A = cannot be the key. & we can't see G in any of the FDs. neither in LHS Nor in RHS and also in any of RHS → E also not present.

So E & G = must be the part of our Candidate key and only EG = cannot be the key

so combining with A :- AEG^+ = {A, E, G, B, C, D} ✓

Yes this is our reqd candidate key.

Also :- we can replace A with D in AEG :-

DEG^+ :- $\{D, A, E, G, B, C\}$

also replacing A with B in AEG

$BEG^+ = \{B, E, G, C, D, A\}$ ✓

Hence AEG, DEG and BEG are our candidate keys.

Lesson 14#

20/08/2022.

Qu consider the relation R(A, B, C, D, F, G)
and FDs = { BCD → A, BC → E, A → F, F → G, C → D, A → G }
then find the minimal set.

Soln. Here we can remove A → G because :
A → F and F → G is given so it is understood that A → G ✓

Hence (A → G) → removed from given set of FDs.
↳ a redundancy → Jiski Jaroorat Nahi Hai!

and BCD → A and BC → E
combinly we can write :-
BC → EA ✓

So in minimal set :- { BC → AE, A → F, F → G, C → D, }

Now from these, we can derive all above given FDs.

Normalization :- process of minimizing redundancy from a relation or set of relations.

redundancy means :-
multiple values/records are placed over multiple locations.
→ record duplication or repeated values

→ so when you design a DB schema, and then accordingly you insert the values / records and the columns / attributes you selected → they may contain those values which are repeated many times → so later on during updation or deletion or insertion → you may face problems.

↳ So to eleminate those problems: we apply the Normalization.

→ So:- Redundancy in relation may cause insertion, dletion and updatt anomalies

↓

means problems !

→ Now Understanding ỷ Bad DB design we have, then what kind of problems we can face:-

So Now Consider the following table:-

C_Id	C_name	P_Id	P_Name	P_Price
C1	Chatur	P1	Shirt	1500
C2	Suhas	P1	Shirt	1500
C3	Rencho	P2	Jeans	2200
C4	Piya	P4	Trouser	1900
C5	Piya	P3	Tshirt	1300
C6	Suhas	P4	Trouser	1900
C7	Viru	P1	Shirt	1500
C8	Raju	P3	Tshirt	1300
C9	Viru	P1	Shirt	1500

→ Insert Anomalies :-

Now here in our table if we want to insert a New product :-

P5	shorts	999

and No any customer has bought it so the C_ID and C_Name = NULL

But we cannot keep these two attributes as Null and also we want to insert a new product.

so here the Insertion anomaly occurs becoz of the Bad design of DB :- customer table and product table should be seperate But here Both are merged.

Hence → Inserting a new product w/o any customer purchase will not be feasable.

→ Update anomalies :-

Now let's say we want to update the trouser's price to 2100. then the Balancing error will come :- money is collected Based on previous pricing and now the price has ↑ed so the remaining Balance amount =? where it gone.?

↓

This Balancing problem will come.

→ delete anomaly :- Now let's say we want to remove/delete the trouser from store :- So in our DB :- we'll remove the trouser from the products :-

and because of Bad DB design :- if we remove the trouser's records :- then the corresponding customer will also be deleted. who had purchased Trousers and all other products.

↳ This is what delete anomaly is

Now these anomalies is catched by Normalisation process !

↳ So Normalization will reduce those redundancies and it will make the DB designs in such a way that such kind of anomalies will be reduced.

→ So the Normalization is the process of reducing or minimizing the redundancy from a relation or set of relations.

So ultimately :→ One particular table or relation should be Based on one theme only.

means one table should contain only one type of related content :- like only customers info

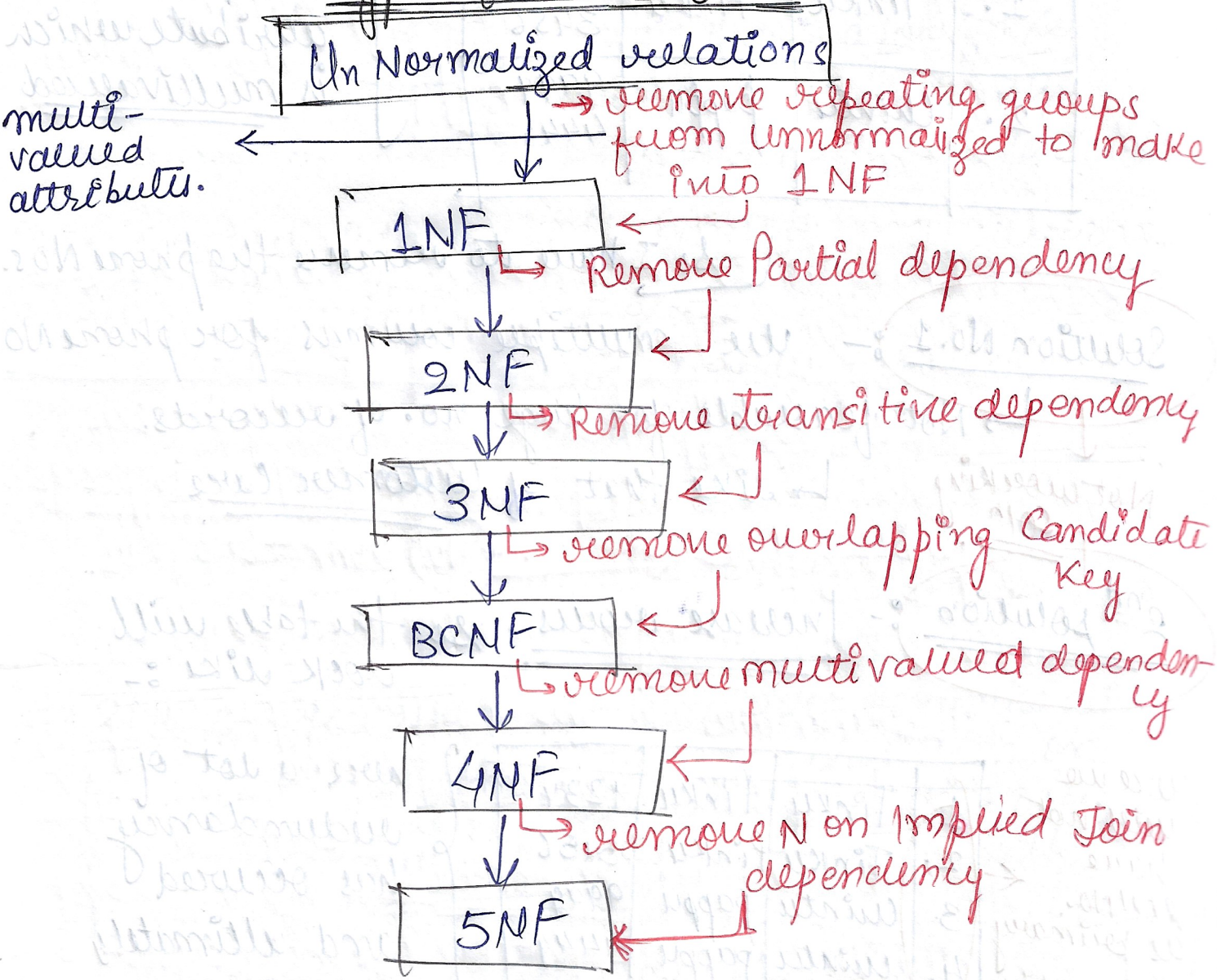
↳ in another table :- products info

↳ — " — :- customer purchase info.

→ if that is not present :- then Normalisation will catch it & make it accordingly.

→ Normalization will break the table into multiple tables with one-one theme only!

⇒ Types of Normalization :-



#1NF :- A relation R is said to be in 1NF if No any Multivalued attribute in R.

Student

RNO.	Name	F-Name	Phone
1.	Tinku	Pinku	1234 3456
2.	Chintu	Pappu	9999 4444

} Now this table has "phone" attribute which is multivalued.

↳ So I have to remove the phone Nos.

Solution No. 1 :- use multiple columns for phone No.

↳ Not feasable for huge no. of records.

↳ like that of customer care

↓ Not working soln

2nd solution :- Increase rows. so the table will look like :-

Here we will not have Roll No. as primary key which we wanted

1.	Tinku	Pinku	1234
2.	Tinku	Pinku	3456
3.	Chintu	Pappu	9999
4.	Chintu	Pappu	4444

} here a lot of redundancy has occured and ultimately we have to remove or reduce redundancy & not ↑ se it.

So ultimate solution :-

3rd Solution : remove "phone No." from student relation & keep it in another relation with primary key of student.

So Now here in 3rd solution :- we will divide the student table into two parts

Student		
1.	Tinku	Pinku
2.	minni	pappu

Contact	
RollNo.	Phone No.
1	1234
1	3456
2	9999
2	4444

↳ This is a foreign key which will refer to the primary key of student table.

Qu. consider Relation R(A,B,C,D,E). If records & details of attributes → Not given

↳ then we will consider the relation R is already in 1NF

↳ almost all questions will come from 2NF onwards

↳ they will already have 1NF !

2NF :- for 2NF :- first we will normalise all the tables till 1NF and then on each table → individually we will apply the rule of 2NF !

↳ assume :- relation R(A,B,C,D,E)

we have composite key = CD and

So prime attributes = C and D

& Non-" " = A, B & E.

→ Now what is partial dependency :- any subset of Key ↓ any prime attribute can derive Non prime attribute.

So in our example :- 'C' = can derive any of :-
A, B or E.

Similarly D also can derive A, B or E
then we will have the partial dependency.

$C \rightarrow A$ or $C \rightarrow B$ or $C \rightarrow E$
$D \rightarrow A$ or $D \rightarrow B$ or $D \rightarrow E$
} these all are partial dependencies.

Partial dependency !

Ex ③ $R(A, B, C, D, E)$ } Now here we cannot have any partial dependency
Now C.Key = C
↓
only prime attribute

because No any proper subset of C is possible

Subset of C ← Null cannot be a key.

Ex 4 $R(A, B, C, D)$
with C.Key = [A, D] → Both Keys are separate.
↳ here also No any partial dependency because of same reason as above

Ex ② $R(A, B, C, D, E)$ Key = ABC → prime = A, B, C
→ Non prime = D, E.

then following are the partial dependencies :-

$A \rightarrow D$	$A \rightarrow E$	$AB \rightarrow D$	$BC \rightarrow D$	$AC \rightarrow D$
$B \rightarrow D$	$B \rightarrow DE$	$AB \rightarrow E$	$BC \rightarrow E$	$AC \rightarrow E$
$C \rightarrow D$	$C \rightarrow E$			

→ Now if all attributes of a relation are prime :-
then also No any partial dependency !

defn of 2NF :- A relation R is said to be in 2NF if it is already in 1NF and there is No any Non prime attribute in R which is partialy dependent on prime attributes of R.

↳ means No any partial depen. + already in 1NF. should be

Ques R(A, B, C, D)

FD = {AB → C, A → D}

↳ Now here Candidate Key = AB.

AB^+ = {A, B, C, D}

prime = A and B.

Non prime = C, D.

Now check for 2NF → (i) it is in 1NF or Not? ↳ we will consider to be in 1NF

↳ (ii) partial dependency ??

↓

Yes present :- [A → D]

Now we have to eliminate this.

So we will remove the column 'D' from this table R :- and will move into another table.

↳ This is decomposition.

remove the column 'D' from original relation and keep it in another relation with A because D is partially dependent on A.

Now after this decomposition :-

$R(A,B,C)$ and $R_1(A,D)$

$FD = \{AB \to C\}$ $\quad$ $FD = \{A \to D\}$ → distribution of FDs also will be done.

Key = AB $\quad$ Key = A

Now no any partial dependency in any of the table.

Que. Consider $R(A,B,C,D)$ with C.Key = AB. Select the FD which will make the relation not in 2NF?

$AB \to C$, $AB \to D$, $\boxed{A \to D,\ B \to C}$

So we will be having following three relations :- Partial dependencies → Now removing it

$R(A,B)$ $\quad$ $R(A,D)$ $\quad$ $R(B,C)$

↳ Original table left with A,B only

Que. $R(A,B,C,D,E)$ $\quad FDs = \{AB \to C,\ D \to E\}$

Soln Then here the C.Key = ABD

$ABD^+ = \{A,B,C,E,D\}$

Then partial dependencies = $\boxed{AB \to C,\ D \to E}$ ✗

Now removing these FDs

$\boxed{R(A,B,D) \text{ and } R_1(A,B,C) \text{ and } R_2(D,E)}$

↳ Now in 2NF

$\boxed{AB \to C}$ $\quad$ $\boxed{D \to E}$

22/08/2022

lesson 15#

#3NF :- A relation R is said to be in 3NF if it is already in 2NF and there is No any Non-prime attribute in R which is transitively dependent on the key of R.

↳ Non prime attribut transitively dependent Nhi hona chahaiy on any key.

So will search like :-

Key → np1

np1 → np2

Key → np2 } Nonprime key 2nd is transitively dependent on key.

Ex R(A,B,C,D)

FDs = {AB → C, C → D}

Key :- $AB^+ = \{A, B, C, D\}$

Key = AB. Non prime C → D

and No any partial dependency here :- Hence the relation is in 2NF.

→ Now any transitive dependency :-

AB → C

C → D

AB → D } D is transitively dependent on key AB.

Now, we have to eliminate it.

decomposition :- remove D from original relation & keep it in another relation with C.

R(A,B,C) $R_1(C,D)$

FD = AB → C $C \rightarrow D$

Que- consider R(A,B,C,D,E,F) , FDs = {A → BCF, C → DE}

Now Normalize this Relation till 3NF.

↳ firstly we have to Normalize to 2NF :- then whatever tables we got from 2NF, in each and every table we have to apply the rule of 3NF.

Key here :- (A) = only c-Key.

$A^+ = \{A, B, C, F, D, E\}$ ✓

Now No any partial dependency.

Hence Now :-

A → BCF

So A → C

C → DE

Hence A → DE } D & E are transitively dependent on A → Key

Now decomposition :-

R(A,B,C,F) , $R_1(C,D,E)$

FDs :- A → BCF , C → DE

Que. Consider R(A,B,C,D) with C.key = AB. Select the FD which will make the relation Not in 3NF?

$AB \to C$, $AB \to D$, $C \to D$, $D \to C$.

Both key C & D are transitively dependent on AB

Que. $R(A,B,C,D,E)$, $FDs = \{AB \to C, A \to D, C \to E\}$

Soln. Key = AB

Now here we have the partial dependency :-

So decomposition acc. to 2NF => $R(A,B,C,E)$, $R_1(A,D)$

Now in each R and R_1 = checking for 3NF.

So No any transi. depen. on R_1

and in R :- $R(A,B,C,E)$

$FDs = \{AB \to C, C \to E\}$ ✓

Yes :- E is transitively dependent on AB.

decomposition :-

$R(A,B,C)$	$\{$	$R_1(A,D)$,	$R_2(C,E)$
FDs :- $AB \to C$		$A \to D$	$C \to E$

BCNF (Boyce Codd Normal form)

↳ A relation R is said to be in BCNF, if it is already in 3NF, and for every functional dependency $\boxed{\alpha \to \beta}$, α must be SuperKey in R.

↳ means every FD of the given relation should have the LHS = a Key.

Ex $R(A, B, C, D)$, $FDs = \{AB \to CD, D \to B\}$

Now Key = $AB^+ = \{A, B, C, D\}$

↳ C.Key. = AB

also AD also a Key

Non prime attri. = only C.

→ partial dependency :- prime → Non prime.

Now $D \to B$ → Not a PD.

↳ B :- a prime attribute

↳ Not present

→ trans. depen :- No any Non prime attribute is in TD

↳ $D \to B$ → TD on prime attributes.

→ Now BCNF :- for FD : $AB \to CD$ } AB is Key

: $D \to B$ } D = alone not a key.

↓

Not in BCNF !

now making in BCNF :- decomposition :-

remove B & keep it in another relation.

(B :- Not dependent directly on Key)

So:- R (A, C, D) & R1 (B, D)

FD:- ~~ABCD~~ A → CD

FD = {D → B}.

Here alone A have to be the key.
as original it was AB in LHS.
& ~~Hence~~ the original FD [AB → CD] is not preserved.
Hence this is not a good decomposition.
Since 'B' ~~is~~ Jo Hai vo galat tarike se dependent Hai 'D' pe.

→ Questions will be like :-

→ a relation will be given & we have to determine that the given relation is Normalised upto how many levels? → till 1NF or 2NF or till 3NF or till BCNF ?

one by one we have to check. ↵

→ So what is the Max^m Normal form is in this relation R.

2NF :- check for P → NP :- P = proper subset of key.

3NF :- Key → NP1
NP1 → NP2
Key → NP2 ⇒ if yes :- then Not in 3NF.

BCNF :- [Key →] → anything.

Que. consider relation R(A, B, C, D, E, X & Y)
and FDs = { AB → C, AC → B, AD → E, E → X,
B → D, BC → A }
then highest possible NF satisfied?

Sol" Here Key = ABY

$(ABY)^+$ = { A, B, Y, C, D, E, X }

then ACY :- also a Key
also BCY →

} prime attributes :- A, B, C, Y

Non prime :- D, E, X.

Now here we have FD :- B → D

Hence the relation R :- not in 2NF

Hence R = max^m 1NF satisfied.

Now if B → D was not any FD :-
then relation R :- satisfies max^m which NF?

Key :- ABY, ACY & BCY

then here in FDs :- E → X is present

$NP_1 \rightarrow NP_2$

then Definitely Not in 3NF.

Hence max^m 2NF only !

* Note :- (1.) if No any Non prime attribute is present :- then the relation 'R' is already in 3NF!

(2.) 3NF = a this Normal form is considered adequate for normal relational DB design.

→ do it till here and BCNF also then the DB design is considered really good!

Ex. Consider a rel^n R(A, B).

3 possible keys = A, B and AB.

A → B B → A → should be there.

So this relation is already in BCNF.

(3.)

→ A relation with 2 attributes is always in BCNF

↳ also klas Binary relation. !

23/08/2022

Lesson 16

Qu R(A, B, C, D, E, F)

FDs = { A → BC, BC → D, D → EF, EF → A }

decompose till 3NF

This is like a cyclic dependency kind of thing.

Key :- A^+ = { A, B, C, D, E, F }

→ C.Key = A, BC, EF, D.

So here all are prime attributes !

and No any Non prime attribute

Hence the relⁿ is already in 3NF !

So when you have a cycle :- then there will be No any Non prime attribute.

Qu. assume R(M, N, O, P, Q) is already in 3NF. then which of the following 3FDs will support the relⁿ :-

A. MN → O B. MO → Q C. MN → P

D. OP → Q → This will result a transitive dependency.

Soln

So key should be = MN.

MN^+ = { M, N, O, P, Q }

prime attri. M is involved.

So here No any Partial dependency & — " — transitive — " — } we will not allow this.

→ so prime attribute = M & N

Non prime — " — = O, P & Q

then FD :- → OP → Q → Not allowed

completely Non prime ← NP → NP

Qus. Consider: dept Sales (dept No., dept Name, month, year sales)

and consider FDs = { dept No. → dept Name,
dept No. Month Year → sales }

The relation Supports from?

Soln. Key :- dept No. Month Year

Now the FD :- dept No. → dept Name
is the Partial dependency
so this rel^n is not in 2NF!

rel^n dept Sales is normalized till 1NF.
then redundancy is there.
and due to redundancy :-
Insertion anomaly, delete anomaly,
update anomaly & inconsistency
could be the problem.

*upto 3NF :- most of the problems are gone.
↳ adequate & self sufficient!

---

lossy v/s lossless decomposition :-

→ lossy Join decomposition :- The decomposition of relation R into R1, R2, R3 --- Rn is lossy when the join of R1, R2, R3, --- Rn does Not produce the same relation as in R

→ So Basically we decompose 'R' into multiple relations then humne un sabka Join Nikala :- Join will be the Inner Join. → then Inner Join karne ke baad hume same 'R' milna chahaiye aur agar same 'R' Nahi Mila :- toh Voh Lossy Join decomposition hoga

Ex: R → Student

RNo.	Name	dept
1	TP	CSE
25	TP	AIML

decomposed into following two tables :-

R_1 → Student

RNo.	Name
1	TP
25	TP

R_2 → dept_details

Name	dept
TP	CSE
TP	AIML

matching column

Now Join of R_1 and R_2 :-

RNo.	SName	dept
1	TP	CSE
1	TP	AIML
25	TP	CSE
25	TP	AIML

R' → 4 records here

Now R and R' are Not Same

↓

Hence this is Lossy Join decomposition.

→ Lossless Join decomposition :-

The decomposition of relation R into $R_1, R_2, R_3 \ldots R_n$ is lossless when the Join of $R_1, R_2, R_3, \ldots R_n$ produces the same relation as in R.

Ex :-

Student = R.

RNO.	Name	dept
1	TP	CSE
25	TP	AI

decomposed into R_1 and R_2 :-

R_1 = Student

RNO.	Name
1	TP
25	TP

R_2 = dept.

RNO.	dept.
1	CSE
25	AI

→ matching column for cond^n of Join on RNO.

then the Join of R_1 and R_2 :-

R_1 inner Join R_2 :-

R' =

RNO.	Name	dept
1	TP	CSE
25	TP	AI

Now R' and R Both are Same

↓

Hence this is lossless !

→ so loss here means :- some extra info came which we don't Need at all !

→ assume decomposition of relation R into $R_1, R_2, R_3 \dots R_n$. Then :-

↳ $R = R_1 \bowtie R_2 \bowtie R_3 \bowtie \dots \bowtie R_n \Rightarrow$ lossless

↳ This symbols denotes :- the Natural Join in relational algebra.

↳ $R \subset R_1 \bowtie R_2 \bowtie R_3 \bowtie \dots \bowtie R_n \Rightarrow$ lossy

Qu: Consider R (X Y Z) :-

R

X	Y	Z
1	5	4
2	5	2
1	8	6

The decomposition of R into 2 relations :- $R_1(X, Y)$ and $R_2(Y, Z)$ is lossless or lossy?

Solⁿ:

R_1

X	Y
1	5
2	5
1	8

R_2

Y	Z
5	4
5	2
8	6

Now R_1 inner Join R_2 :- on common column : Y

$R' = R_1 \bowtie R_2$

X	Y	Z
1	5	4
1	5	2
2	5	4
2	5	2
1	8	6

We are not getting the original relation R

Hence lossy

→ A relation R decomposed into two relations R_1, R_2 and attributes in $R_1 \cap R_2 = \alpha$.

↳ common attribute in both.

for lossless decomposition, α should be a Key in either R_1 or R_2. !

Qu. $R(A, B, C, D, E)$, $FDs = \{ A \to BC, D \to CE \}$

decomposition :- $R_1(A, B, D)$, $R_2(D, C, E)$

$FD = \{A \to B\}$ $\{D \to CE\}$

then Lossless or lossy.

Soln Key in R :- AD

$AD^+ = \{A, B, C, D, E\}$

and we have D = common attribute.

Key in R_1 = AD and Key in R_2 = D.

Hence 'D' is key in R_2 :- which is a common col. too.

↓

Hence this decomposition is lossless !

Qu. $R(A, B, C, D, E)$, $FDs = \{ A \to BC, D \to CE \}$ Key = AD

↳ Key = A

decomposition :- $R_1(A, B, C)$, $R_2(A, D, E)$

$FDs = \{A \to BC\}$ $\{D \to E\}$

then lossless or lossy ?

Soln Key in R :- AD

Now common col. in R_1 & R_2 = A

and A = Key in R_1 → Hence lossless

Que. $R(A, B, C, D)$ → Key = A, BC

$FDS = \{A \to BC,\ BC \to A,\ B \to CD\}$

decomposition $R_1(A, B, C)$ and $R_2(B, C, D)$.

Soln. $FDS = \{A \to BC,\ BC \to A\}$ → Key = A, BC

$FDs = \{B \to CD\}$ → Key = B.

Now common column = B & C ⇒ Key in R_1 = BC

Hence lossless.

Now here we have two common attributes B & C then we have to check all the following combinations :-

either one of these cond" is true → then lossless.

or (i) B is key or not
or (ii) C is — " —
or (iii) BC — " —

→ either in R_1 or in R_2

↳ then we found BC as key in R_1.
also B = key in R_2.

dependency preserving decomposition :-

A decomposition $D = \{R_1, R_2, R_3, \ldots, R_n\}$ of R is dependency preserving w.r.t a set F of functional dependency if :-

$$(F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n)^+ = F^+$$

F = all FDs of R.

combining all FDs of decomposed table, we should get (F)

Qus. $R(A, B, C, D)$

$FDs = \{AB \to C, B \to D\} = F$

decomposition = $R_1(A, B, C)$ | $R_2(B, D)$

$FD_1 = \{AB \to C\}$ | $FD_2 = \{B \to D\}$

$FD_1 \cup FD_2 = F$

↓

Hence preserved !

Qus. $R(A, B, C, D)$, $FDs = \{AB \to C, C \to D, D \to A\}$

decomposition: $R_1(A, B, C)$, $FD_1 = \{AB \to C\}$

$R_2(C, D)$, $FD_2 = \{C \to D\}$.

$FD = \{AB \to C, C \to D, D \to A\}$

$FD_1 \cup FD_2 = \{AB \to C, C \to D\}$

we cannot derive $D \to A$

Hence Not preserving.

Qus. $R(A, B, C, D, E)$, $FDs = \{A \to BC, D \to CE\}$

decom:- $R_1(A, B, D)$ $R_2(D, C, E)$

$FD_1 = \{A \to B\}$ $FD_2 = \{D \to CE\}$

Now here we are not getting $A \to D$ in R_1

Hence $A \to C$ is missing in R_1

↳ Hence Not preserving !

24/08/2022

Lesson 17

ER diagram to Relational Model

→ whenever an entity set is given :- from that set you will get a table or relation.

Entity set ⇒ table or relation.

Key ⟶ primary key

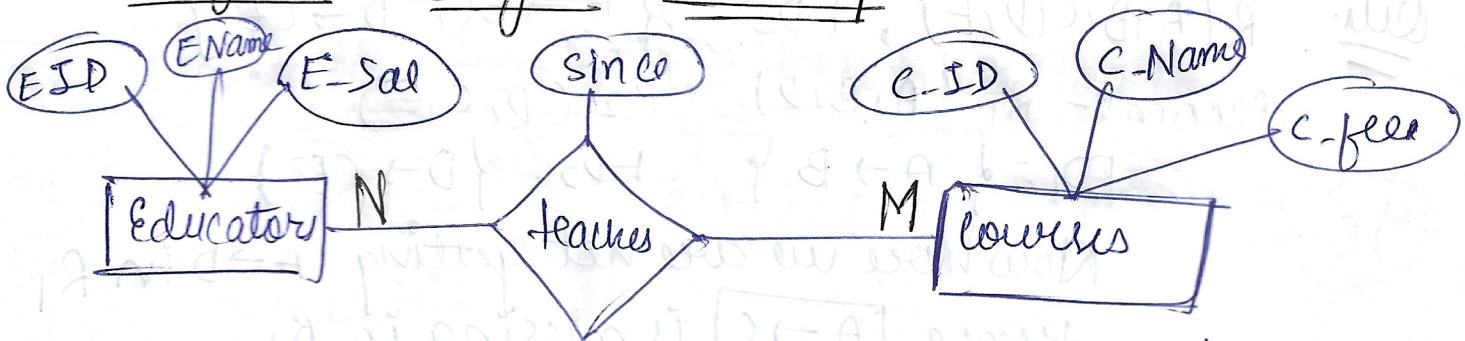
Relation → participation constraint ↔ full participation or partial parti.

Having following two things → mapping cardinalities

- one to one
- many to one
- one to many
- many to many

→ for each mapping cardinalities, their table formation will be different.

→ Many to Many relationship :



Now for each entity set :- we will be having one particular table

Tables : 1. ~~Edit~~ Educator (EID, E_Name, E_Sal)
P.Key

2. courses (C_ID, C_Name, C_fee)
P.Key

Now for the relationship set = teaches, we have following three possibilities :-

1. either add in the Educators table
2. or add in courses table
3. or have a separate table for "teaches" rel^n.

But we Need to check out Based on Current Scenario about many to many, which possibility is good for teaches relationship among above 3 possibilities !

1st :- when rel^n mentioned in educators table :-

Educator

E_id	E_Name	E_Sal	C_id	Since.
1.	VD	10K	C1	2007
1.	VD	10K	C2	2008

Here one educator can teach multiple courses :- hence for each course one entry → So here the E_id will not be a primary key anymore.

↳ repeated values for key ⇒

Hence this is Not a good idea.

Similarly when we will keep this rel^n in courses table :- same problem will come.

course

C_id	C_Name	C_fee	E_Id	since
C1	COA	5K	1	2011
C1	COA	5K	2	2015

→ Here also C_Id is not unique. Hence it will be No more a Primary Key

↳ So here also redundancy & inconsistency problems may occur.

→ so previous two were not a good idea.
Hence we will go with a separate table for teaches :-

teaches (E_id, C_id, since)

E_id, C_id → foreign key constraint

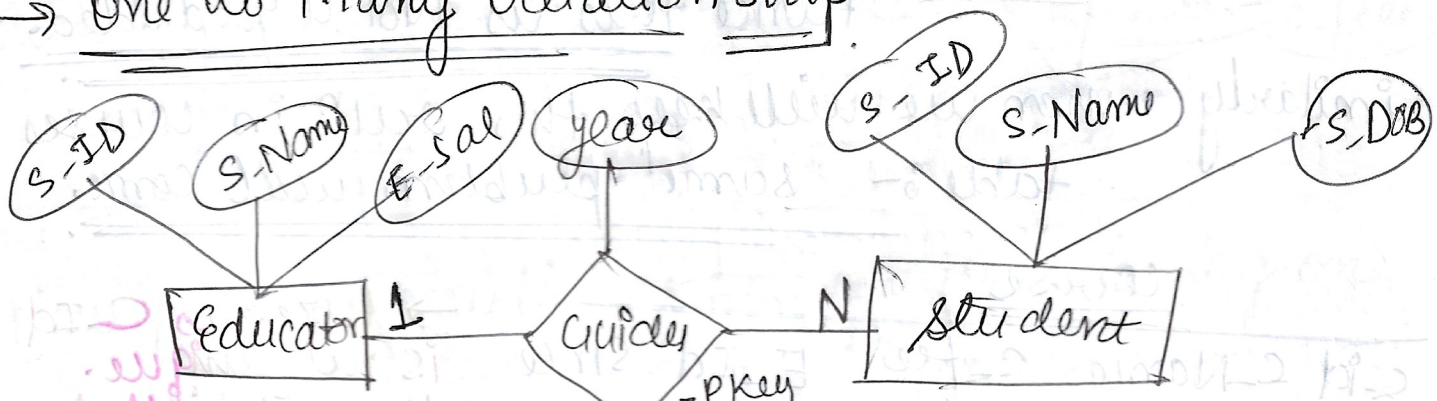
E_id, C_id → primary keys of both table only

→ and rest of details can be fetched from their respective tables!

Here combination of E_id and C_id compositively will be a Key.

→ so for many to many rel^n :- three separate tables are needed.
- two for two entity sets
- one for rel^n

→ One to Many relationship



tables :- Educator (Eid, EName, E_Sal)
(Eid: PKey)
Student (S_ID, SName, S_DOB)
Guides (EID, SID, year)
(SID: PKey)

→ Here one educator can guide many students in same year :- then we will have multiple redundant data in guides table. we will have multiple similar EID entries with diff. SID. in guides table.
↳ SID will always be unique.

Now 2nd option : Only two tables :-

1. Educators (EId, EName, ESal)
PKey.
↳ Here we cannot keep the relation guides in Educators table :- since one educator guides multiple students → so EId will have redundant values → Hence Not a good idea.

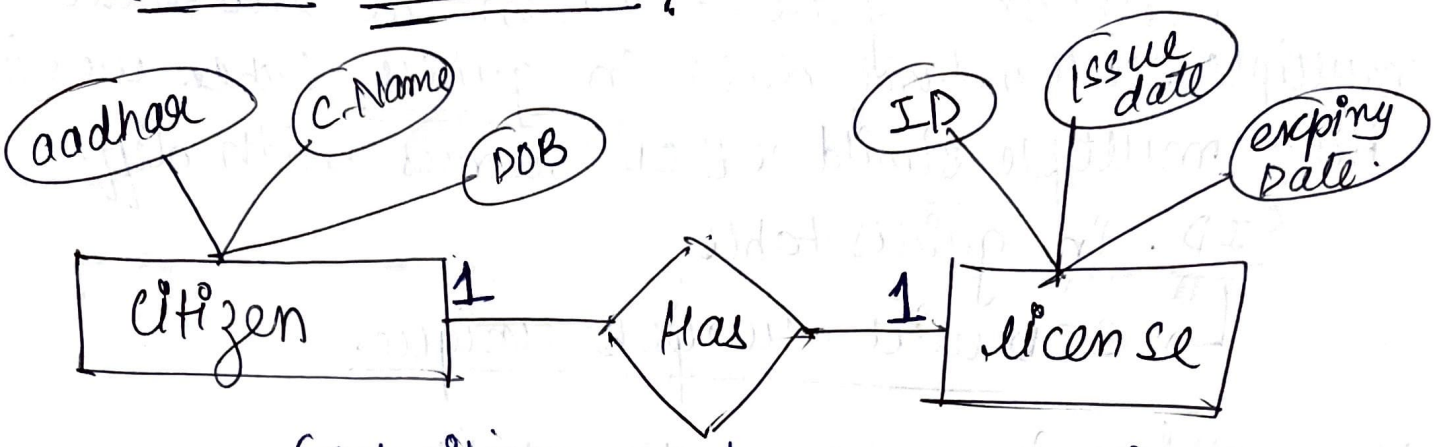
2. Student (SID, SName, SDOB, Eid, year)
PKey.
↳ Here each student will have only one educator to guide him. Hence SID still will have all values unique and No any redundant values.

Yes, this can be a good option of only two tables !!

→ So whichever many relationship we have :- will keep the relation over there.
↳ (Not for many to many).

for total participation of student entity make Eid here "Not NULL"

→ One its one relationship



→ Each citizen can have only one license
So here also we have following two options :-

1st : Citizen (aadhar, Name, DOB, License ID)
License (ID, Issue date, expiry date)

2nd : Citizen (aadhar, Name, DOB)
License (ID, Issuedate, expirydate, aadhar)

→ In 1st option : the relation is mentioned in citizen table :- through descriptive attribute : License ID.
and in 2nd option :- the rel" is mentioned in license table.

Both options are valid. But 2nd option is better :-
↓
because there will be total participation of license ID. and there will be less No. of records
Because : every citizen (who is having aadhar) will not have a license → so multiple values will be Null in license ID col. in option 1.
BUT :- every license ID holder will definitely have a aadhar card.

→ So if licence has total participation then option 2 with aadhar ID NOT Null & Unique.

→ in option 2 :-

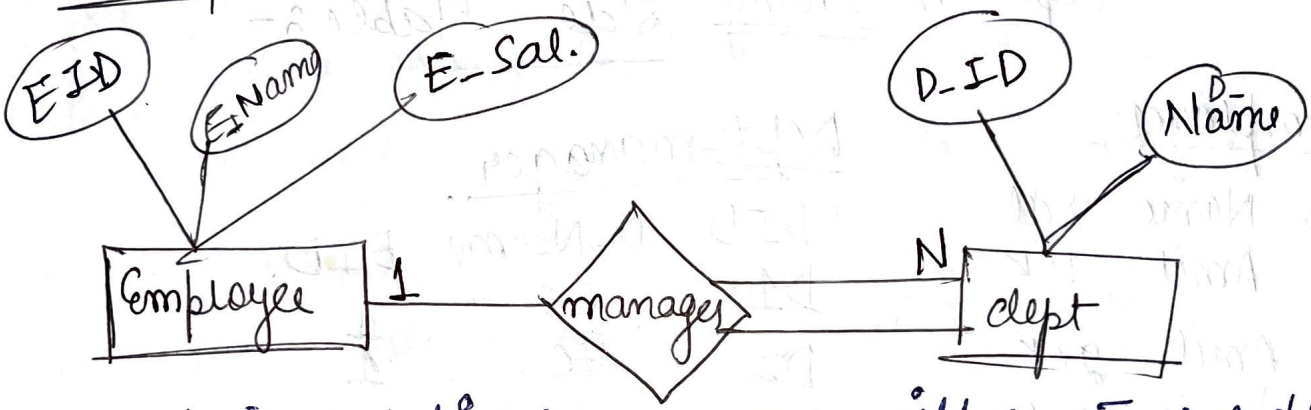
license table :- we have the FD :-

LID → Issuedate Exp.date AadharID.

also: aadhar → LID

and so the rel^n license is in BCNF!

→ Participation Constraint :-



this relation :- manages will go towards many side.

So option 1 : 3 tables :-

→ Employee (EID EName E_sal).

Employee

EID	EName	E_Sal
1	Amit	10K
2	Amit	20K
3	Samir	30K

manages

Eid	DID
1	D1
1	D2
2	D3

redundant data & both are of foreign key constraint.

dept:

DID	DName
D1	CS
D2	EC
D3	ME
D4	CE

→ as we can see → we don't have the manager for dept D4 → CE → so dept. don't have full participation here.

Now tot al participatn of dept. will be when we have every dept. entry participating in manages relationship.

So here 2nd option comes into picture :-

2 tables with relationship information kept in Many side table :-

Employee

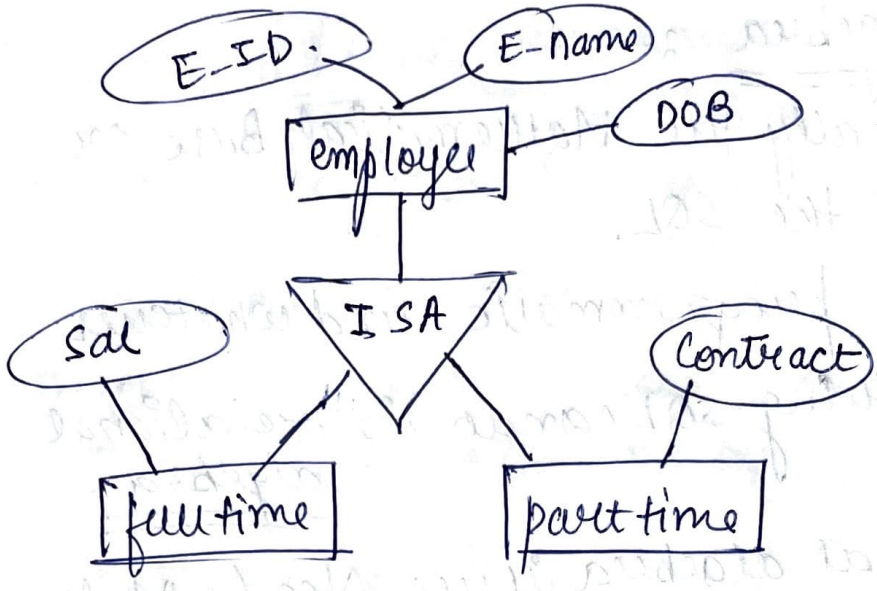
EID	Name	Sal
1	Amit	10K
2	Amit	20K
3	sumit	30K

Dept-manager

DID	D-Name	EID.
D1	CS	1
D2	EC	1
D3	ME	2
D4	CE	3

EID must have "Not Null" constraint. for total participation.

Generalization And Specialization :-



→ 1st option :- → Employee (EId, EName)

here 3 tables → full time (EId, ~~EName~~ Sal)

↳ only primary key of employee table needed. Rest all details are stored.

This option is better for ~~overlapping~~.

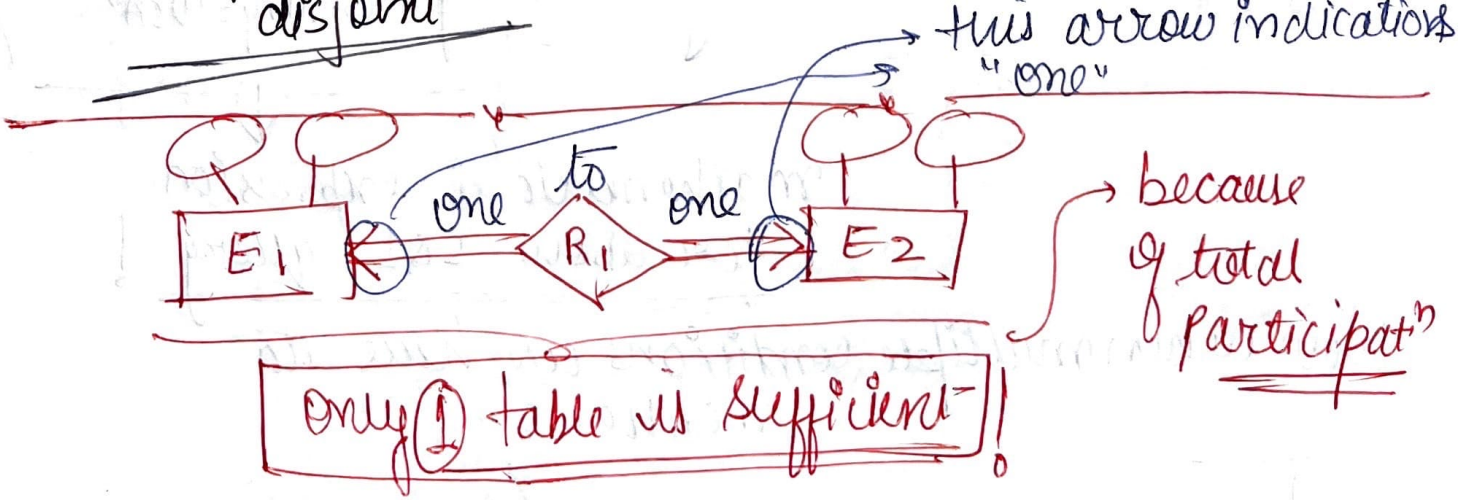
→ part time (Eid, Contract)

→ 2nd option : → Fulltime Emp (EId, EName, EDOB, Sal)

→ part time Emp (EID, Name, DOB, Contract)

↳ two tables only here

This option is good for disjoint



Relational algebra :-

It is Basically the Mathematical Base or foundation for the SQL.

SQL = programmatic and whatever mathematical thing SQL can do is relational algebra.

→ In relational algebra if we Need Math, then we need the following operators :-

1. Select (σ) :- Used to select the Specific rows.

$\sigma_p(R)$:- σ = Select operator
p = predicate or cond^n of where clause
R = relation.

Ex: Select * from customers where country = 'USA'.

So in relational algebra :- $\sigma_{country='USA'}(customers)$

↓

mathematical expression for above SQL query !

Now when multiple conditions we have to mention :-

$\sigma_{country='USA' \text{ and } City='NewYork'}(customers)$.

25/08/2022. #lesson 18#

→ defⁿ of relational algebra :-

it is a procedural query language, which takes instances of relations as i/p and yields instances of relations as o/p.

The main application of relational algebra is to provide a theoritical foundation for relational databases, particularly for query languages.

Operators

01. Select :- it is actually the WHERE clause of SQL. → all attributes / columns will be in result set.

↳ The Select operator chooses those tuples in the o/p that satisfy the specified condⁿ.

written as :- $\sigma_{\text{selection-condition}}(\text{Relation Name})$

Ex: $\sigma_{\text{country}='USA'}(\text{customers})$ → fetches all those customers info who belong from country = USA

→ So Basically Select will filter tuples Based on given condⁿ.

Ex:
Now consider the table = Sailors (Eid, Name, Rating, Age)

Now to select those tuples of Sailors relation where rating is greater than 8

So the relational algebra we will write as :-

$$\sigma_{rating > 8}(sailors)$$

→ comparision operators that can be used in selection conditions are :

$$<,\ <=,\ =,\ \neq,\ >=,\ >$$

→ The select operation is commutative :-

$$\sigma_{cond^n 1}\left(\sigma_{cond\,2}(Rel^n)\right) = \sigma_{cond\,2}\left(\sigma_{cond\,1}(Rel^n)\right)$$

from here we will get one result set.

and in this resultant table :-

we will apply this cond^n 1.

Ex: Now, in above Sailors table :-

if we write :-

$$\sigma_{age > 25}\left(\sigma_{rating > 8}(sailors)\right)$$

↳ this will fetch us all those records who are older than 25 years and have rating > 8.

Now, $\sigma_{cond^n 1}\left(\sigma_{cond\,2}(Rel^n)\right) = \sigma_{cond^n 1 \wedge cond^n 2}(Rel^n)$

And operator can also be used.

→ AND = $\wedge$

→ OR = $\vee$

Qu. find all customers details from rel" customers who are from country USA or UK

Soln. So the expression :-

$$\sigma_{country = 'USA' \vee country = 'UK'}(customer)$$

→ Until Now, whatever we were writing :- that was actually : select * → means all columns :- Now we will be selecting only Specific column's data :-

Some will be Using Project for this :-

02. Project (Π) :- The project operation is used to choose certain columns from the table and ~~trashes~~ out the other attribute fields.

→ A projection is a Unary operation written as :

$$\Pi_{A1, A2, A3, ---, An}(R)$$

Ex : $\Pi_{customerID, country}(Customers)$ → No any filtering in rows

↳ fetches all distinct records of these two columns.

↳ and will not show the repeated ~~records~~ records of same column.

→ Ex:- consider :-

R

A	B	C
1	2	3
1	2	4
1	3	5
2	4	6

$\Pi_{A}R$:-

A
1
2

Now the algebra :-

$\Pi_{A,B}(R)$

will give us :-

R

A	B
1	2
1	3
2	4

Skip the duplicates.

→ So: project provides only distinct tuples by eliminating duplicates from result set.

→ condition mention karne keliye tume abhi Select (σ) operation hi use karna padega

Ex: fetch Name and Salary of employee who have salary greater than 15000.

↳ first we will try to filter the rows based on the given condn. and among those we will give the Name and salary only.

↳ So first we will write select and then will write project :-

$\Pi_{name, sal}(\sigma_{salary > 15000}(employee))$

Ex: write a relⁿ algebra stmt to find Name of all such employees from dept No. = 2 & whose salary is greater than 17000.

Soln $\Pi_{name}\left(\sigma_{dept\ No.=2 \wedge sal>17000}(employee)\right)$

→ Set operations :- 1. Union 2. Intersection (∩) (∪) 3. Set difference (−)
↳ minus / except.

→ The two relations on which set operations are implemented upon must necessarily have similar data types of tuples. this condⁿ is called type or Union compatibility.
↳ also the No. of columns must also be same.

Ex:-

R_1			R_2		
A	B	C	X	Y	Z

So here :- data types of
A = X
B = Y
C = Z

Ex Consider two tables E_1, E_2 having same attributes.
So their union :- $E_1 \cup E_2$ → Vertical Joining of two tables
↓
and this oper will ignore the duplicate rows.
→ This was the direct Union on all the columns.

Now we can also have union on projected / selected columns only :-

Ex:- $(\Pi_{Eid, EName}(E_1)) \cup (\Pi_{Eid, Ename}(E_2))$

→ Similary we can go for Intersection also :- $(\cap)$

↓

common entries will be fetched!

Ex: $(\Pi_{Eid, EName}(E_1)) \cap (\Pi_{Eid, EName}(E_2))$

→ Set difference :- $(\Pi_{Eid}(E_1)) - (\Pi_{Eid}(E_2))$

(→)

This will removes all those tuples from E_1 which are also present in E_2

↓

only unique records of E_1

Now :- $\boxed{E_2 - E_1 \neq E_1 - E_2}$

↳ commutative property Not satisfied in minus

But satisfied in Union and Intersection.

also :- Union & Intersection are associative also.

→ $A \cup (B \cup C) = (A \cup B) \cup C$

→ $A \cap (B \cap C) = (A \cap B) \cap C$

→ Set difference is also Not associative.

Ques. Consider: Students(RNO., Name, DOB) and Employees (EID, Name, Sal.)

write a relational algebra for :-

Select distinct Name from students where dob = '27-10-1988' union select distinct ename where sal > 15000

Soln. $\Pi_{name}(\sigma_{dOB='27-10-1988'}(students)) \cup (\Pi_{ename}(\sigma_{sal>15000}(employee)))$

Ques. Consider $R_1(X, Y)$ and $R_2(X, Y)$. R_1 and R_2 contains all Not Null values. will the following 2 statements be equivalent or Not?

① $\Pi_X(R_1 \cup R_2)$ and ② $\Pi_X(R1) \cup \Pi_X(R_2)$.

Soln.

R_1

X	Y
1	2
1	3
2	4

R_2

X	Y
1	3
2	5
2	4
3	6

then :-

① $\Pi_X(R_1 \cup R_2)$

$= 1, 2, 3$

② $\Pi_X(R_1) \cup \Pi_X(R_2)$

$\Rightarrow 1, 2, 3$

Both are same hence ① = ②.

③ $\Pi_X(R_1 \cap R_2)$: 1, 2 } X

④ $\Pi_X(R_1) \cap \Pi_X(R_2)$: 1, 2 } X

Both are equivalent too in this example.

Now what about :- ① $\Pi_x (R_1 \cap R_2)$ and

② $\Pi_x(R_1) \cap \Pi_x(R_2)$

for :-

R_1

x	y
1	2
1	3
2	3
3	4

R_2

x	y
1	2
1	4
2	3
2	5
2	4
3	6

So ① $\Pi_x(R_1 \cap R_2)$

1
2

② $\Pi_x(R_1) \cap \Pi_x(R_2)$

$\Rightarrow$

1
2
3.

So clearly $\Pi_x(R_1 \cap R_2)$ is Not equivalent to $\Pi_x(R_1) \cap \Pi_x(R_2)$

→ Cartesian product : represented by X.

So $R \times S$ = cross product of R and S :-

R

A	B	C
1	a	b
2	c	d
3	e	f

S

D	E
1	c
2	d

No. of columns in $R \times S$ = 3+2 = 5

No. of rows in $R \times S$ = $3 \times 2 = 6$

→ Cartesian product can be used with projection & also with select :.

$\sigma_{A=D}(R \times S)$

R

A	B	C	D	E
1	a	b	1	c
2	c	d	2	d

Ex: $\Pi_{A,B,D}(\sigma_{A=D}(R \times S))$

A	B	D
1	a	1
2	c	2

→ So any expression in relation algebra will be executed from last (innermost) to first (outermost)

Qu. Consider :- Sailors (sid, sName, age)
Reserves (Sid, boatID, date)

So select Names of all such Sailors who booked boat with id==5.

Soln. We will have to write the condⁿ on common column.

Π_{sname}

$$\left[\sigma_{\substack{(sailors \times Reserves) \\ sailors.id = reserves.id \wedge boatID = 5}}\right]$$

↳ Join condⁿ
↳ for removing the redundant values.

The SQL query corresponding to this :

```
Select distinct SName from Sailors, reserves
where (Sailors.sid = reserves.sid)
AND
(Boat ID = 5)
```

Ex:- find the Name of the customers who have placed atleast one order?

refer customers and orders table, Both having customerID as common.

$$\Pi_{CName}\left(\sigma_{customer.CID = Orders.CID}(customers \times orders)\right)$$

Ex: find the Name of customers who have placed atleast one order shipped by :-
ShippedID = 3

Soln

$$\Pi_{CName}\left(\sigma_{customer.CID = Orders.CID \wedge Shipper.shippedID = 3}(customers \times orders \times Shippers)\right)$$

26/08/2022 #lesson 19#

Only doubts were discussed.!

27/08/2022 #lesson 20#

#Joins :- we have following 3 types :-

1. condition Join
2. Equi Join
3. Natural Join.

→ Natural Join :- Here the Joining cond^n is implicit if two tables have one same column with same column Name :- then go for Natural Join :- $R \bowtie S$

R

c_1	c_2

S

c_1	c_3	c_4

$R \bowtie S \equiv \sigma_{R.C1 = S.C1} (R \times S)$

This is equivalent to

↳ Here we don't have to mention any other condition.

Now we don't want this above cond^n Now, we want to Join two tables Based on some other cond^n. then that Join is Klas [cond^n Join] → Based on some specific cond^n :-

↓

shown/written as :-

$R \bowtie_{condition} S$

Ex:

$R \bowtie S$
$R.C_1 < S.C_3$

Cond^n Join can have :-
$<, <=, >, >=, \neq$

Equi Join :- when there is cond^n of equality in condition Join :- then that is

↓ Klas Equi Join

Naam alag hai But kaam same to same as that of cond^n Join
↳ Bas "equal to" (=) wala sign hona chahiy cond^n me !

Ex:-

$R \bowtie S$
$R.C_1 = S.C_1$

→ and alone $R \bowtie S$ = w/o any condition is Inner Join or Natural Join!

Ex.

Student		
SID	SName	DOB

enrol	
SID	CID

Now I want to have :- which student has enrolled in which courses :-
↳ Just do Natural Join

Student ⋈ enrol = $R \bowtie S$ = that's it

Now Student who has enrolled in CID = 6. :-
↳ Now this will be the conditional Join :-

$R \bowtie S$ } Student ⋈ enrol
cond^n } Student.SID = enrol.SID
CID = 6

↳ Here you have to write all the cond^{n}s

Ex:- consider :-

E

Eid	EName	Rating	Age
20	Rauish	8	24
30	Radha	7	25
40	Shyam	9	26

P

Eid	Pid	Day
20	120	2/8/21
40	115	5/6/21

then the following relational algebra :-

$E \bowtie P$
$E.Eid < P.Eid$

Now this will result into :-

EID	EName	Rating	Age	Eid	Pid	Day
20	Rauish	8	24	40	115	5/6/21
30	Radha	7	25	40	115	5/6/21

*NOTE :- for Natural Join :- the column Names in two different tables Must be same. atleast one-one column in both tables should have same Names.

And when the columnNames are Not same then we have to use the condn Join / Equi Join then the system will print all the columns.

and when you want only specific columns only :- then with Joins as the Inner expression :- you can use projection ($\Pi_{A1, A2, A3, \ldots}$) outside.

→ for Natural Join :-

R

a	b	c	d

S

a	b	e	f

Here two col. are common.

then in Natural Join all the common columns having same column Name will be equated!

So: $R \bowtie S \equiv \boxed{R \bowtie_{R.a = S.a \wedge R.b = S.b} S}$

So Based on these equal columns :- the result will be fetched out.

↓

and same columns will be only once in result set and not multiple times.

Outer Join (OJ) :-
- → Left outer Join (LOJ)
- → Right outer Join (ROJ)
- → full outer Join. (FOJ)

for each of these three Joins :-
- → LOJ = ⟕
- → ROJ = ⟖
- → FOJ = ⟗

consider :-

R

A	B

S

D	E	F

you will have one equality condition.

then LOJ = $\boxed{R \overset{R.A = S.D}{\bowtie} S}$

ROJ :- $\boxed{R \overset{R.A = S.D}{\bowtie} S}$

FOJ :- $\boxed{R \overset{R.A = S.D}{\bowtie} S}$

Ques. find the Name of customers who have placed atleast one order shipped by shipper ID = 3.

You have :- Customers (CID, CName) orders (orderID, CID, shipperID)

Soln. So the reqd relational algebra :-

$$\Pi_{CName}(Customers \bowtie orders \; customers.CID = orders.CID \wedge shipperID = 3)$$

or using cross product :-

$$\Pi_{CName}(\sigma_{customers.CID = order.CID \wedge shipperID = 3}(customers \times orders))$$

Ques. find all such drivers name who have driven blue colour car.

Given :- Drivers (Did, DName, Rating)
Cars (CID, cmodel, color)
Drives (DID, CID, dateOFRace)

Soln. $\Pi_{DName, drivers.Did}(Drivers \bowtie Cars \bowtie Drives \; Drivers.DID = Drives.DID \wedge Cars.color = Blue \wedge Cars.CID = drives.CID.)$

→ using cross product & σ :-

$$\Pi_{DName, drivers.Did.}(\sigma_{color = 'blue' \wedge Cars.CID = drives.CID \wedge Drivers.DID = drives.DID}(Drivers \times Cars \times drives))$$

1st condn = Color = 'blue'

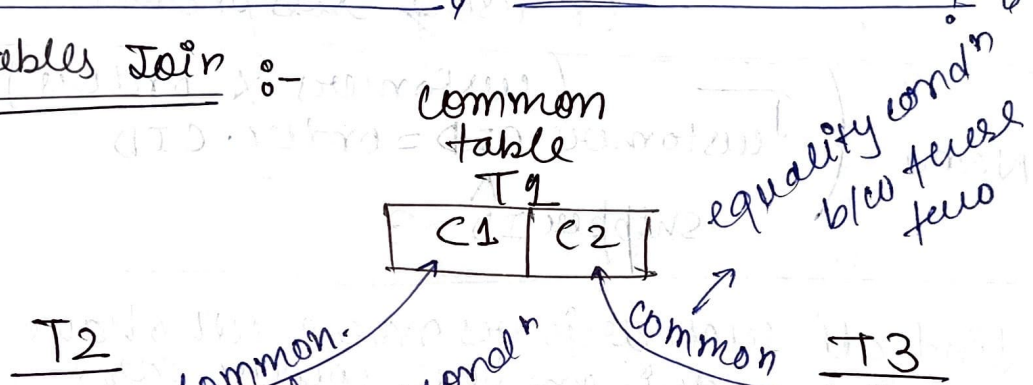
2nd :- blue colour's CID we will have to match in drives table to check if someone is driving that car or NOT !

↳ [Cars.CID = Drives.CID]

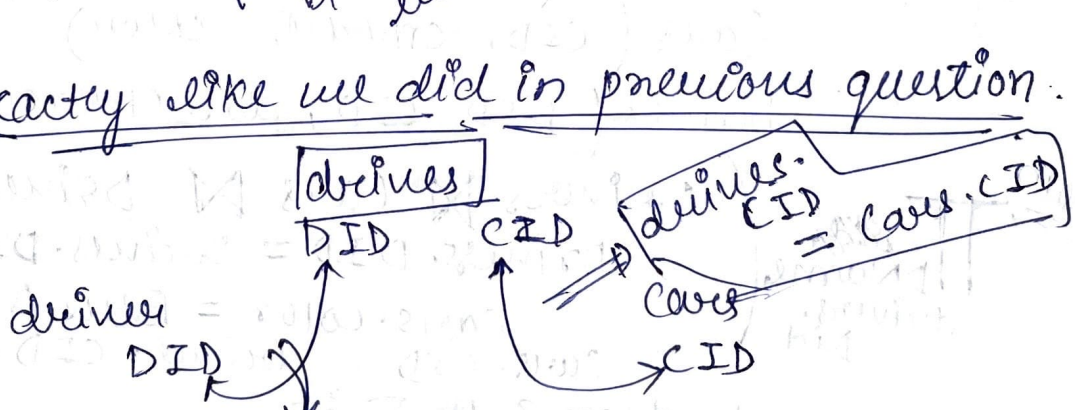
3rd condn :- Drives.DID = Driver.DID.

↓ These 3 condns must be written in Select !

→ 3 tables Join :-



→ exactly like we did in previous question.



Qu write a query to find all such drivers Name who have driven blue color car or Black color car.

Soln

$$\Pi_{\substack{drivers.Did,\\ DName}} \left[\sigma_{(color = 'Black' \vee colour = 'Blue') \wedge (cars.cid = drives.cid) \wedge (driver.did = drives.did)} (drivers \times cars \times drives) \right]$$

Qu. Now, find all such drivers Name who have driven blue color car AND Black color car.

↳ those drivers who have driven Both colour cars.

↳ Here we need to have intersection of two queries.

- 1st query :- drivers who driven Black colour
- 2nd query :- —— " —— Blue —" —

↳ intersection of these two queries will gives us those drivers who have driven both.

Soln.

$$\Pi_{\substack{drivers.Did,\\ DName}} \left[\sigma_{driver.Did = drives.Did \wedge cars.cid = drives.cid \wedge car.color = 'Blue'} (drivers \times Cars \times drives) \right]$$

$\cap$

$$\Pi_{\substack{drivers.Did,\\ DName}} \left[\sigma_{Driver.Did = drives.Did \wedge car.cid = drives.cid \wedge car.color = 'Black'} (drivers \times Cars \times drives) \right]$$

Rename Operator :- Symbol : ρ = rename

↳ Same as that of aliasing of SQL !

So if you want to change the table :- Student's Name to S :-

$$\rho_S (Student)$$

Now renaming of following table :-

Student(RNO, SName, DOB)

$$\rho_{Stud(R,S,d)} (Student)$$ → This will change the table's Name as well as the column's Name

and when you want to change only column Names :-

$$\rho_{(R,S,d)} (Student)$$

and renaming a specific column :

$$\rho_{SName \to S} (Student)$$

or

$$\rho_{S \leftarrow SName} (Student)$$

29/08/2022 # Lesson 21 #

#division Operator :- so for two relations to be divided :- $R_1 \div R_2$ following is the condition :- $R_2 \subseteq R_1$

↓

i.e. attribute set in R_2 must be the subset of attribute set in R_1.

R_1		
x	y	z

R_2	
x	y

} Yes this will work.

attribute set of R_1 = {x, y, z} } $R_2 \subseteq R_1$

— // — R_2 {x, z}

Result set of $R_1 \div R_2$ = will have attributes which are present in R_1 but Not in R_2

↓

attribute Set of R_1 − attribute set of R_2

{x, y, z} − {x, y} = {z}

Now which rows of column 'z' will come?

↳ that value of col. z :- which is associated with every row of R_2 relⁿ.

R_1		
x	y	z
A_1	A_2	C_1
A_4	A_5	C_2
A_1	A_2	C_2
A_4	A_5	C_4

R_2	
x	y
A_1	A_2
A_4	A_5

A_1 A_2 → C_2

A_4 A_5 → C_2

→ Here the value C_2 = is associated with all the rows of relⁿ R_2

Hence the result set of $R_1 \div R_2$:-

z
C_2

Ex: roll No. of students who have enrolled for all courses :-

R.No.	CID
1	1
2	3
1	2
1	3
2	2
3	1

CID
1
2
3

Now doing the following thing we will get our req^d ans :-

$$\frac{\Pi_{\text{roll No., CID}}}{\Pi_{\text{CID}}}$$

roll No. of student who has enrolled for all courses.

* and if No one is enrolled for all the courses :- then the division will give you :-

NULL !

Que. consider :- Cars (cid, model, colour)
Drives (did, cid, date)
so write a query to find all such drivers id who have driven all cars in a day :-

Soln.

$$\frac{\Pi_{\text{did, cid, date}}(\text{drives})}{\Pi_{\text{cid}}(\text{cars})}$$

Ans.

if $\frac{R1}{A\ B\ C}$ and $\frac{R2}{D\ E\ F}$ then $R_1 \bowtie R_2 \equiv R_1 \times R_2$

↓ cross product

because No common attribute in R_1 and R_2

Ques. consider student (name, sex, marks)
then the relational algebra gives what :-

$\Pi_{name}(\sigma_{sex=female}(student))$ – $\Pi_{name}(student \bowtie_{student.n, x, m} (sex = female \wedge x = male \wedge marks \leq m))$

Part 1. Part 2.

Soln So here we have two queries :-

So Part 1 denotes : list of all female students.

& Part 2 : → Names of female students having lesser marks than atleast one male student

Student		
Name	Sex	Marks
n_1	F	20
n_2	F	21
n_3	F	26

Student		
n	x	m
n_4	M	21
n_5	M	21
n_6	M	25

then in result set of Part 2 :-

n_1	F	20
n_2	F	21

↳ so these two will be selected.

→ So the answer will be :-

Name of all those female students with more marks than all the boy students.

* discussed the PYQs of Relational Algebra.

30/08/2022.

Lesson 22

* discussed PYQs of Relational Algebra and of SQL.

! that's it !

01/09/2022.

Lesson 23

* discussed PYQs of SQL !