Lec (7) → Schema Refinement

DM

Normalization

Logical DB Design is not very good — as — Conceptual modeling is subjective Process

Thus

for this → we need to refine it. or ensure it is good

a formalism to do → Normalization comes

this

Relational DB Schema → set of relations

set of attributes

How we group the attributes to relations is very important.

Normalization = schema refinement → help determine good relations.

Purpose of Normalization

- > Avoid redundancy
- > relational principle conformed data
- > more able to change
- > Avoid certain updating 'Anomalies'
- > intergrity constraints - facilitate to enforce

Redundancy and Data Anomalies

(*) The redundant data could be removed without the loss of information

NO	Job	dep No deg	dep name dname	dep city city
10	Sales	10	Sales	Kandy
51	Mng	20	Acc	Colombo
40	Clerk	40	Acc	Colombo
45	Clerk	30	operati	Colombo
46	Clean	40	Cleanig	Galle

Department information table(s) ↓ and

These are repeating

anomalies (issues) → happen coz redundancy

Redundancy → Disk space

Anomalies

↳ issues

Insert Anomaly

Department info
- dep no (key)
- dep name
- dep city

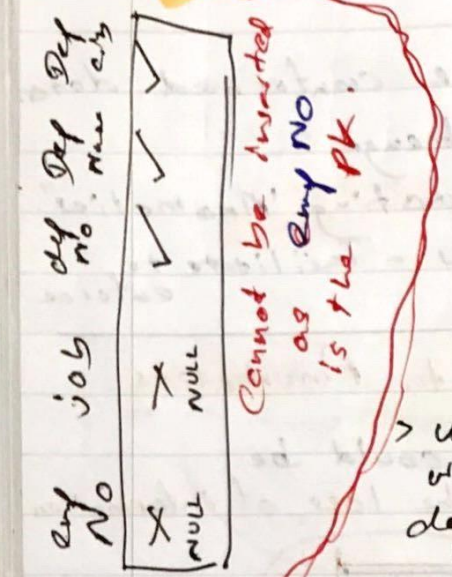
are ready

But we have to add those data with

↓

an employee.

> until we receive an employee we can't add data of department → dep data depend on employee



—	—	~~20~~	Acc. Berlin
—	—	20	Acc Berlin
—	—	30	Oper. Berlin

← The same department data (redundant)

Update Anomaly

→ To update this we need to update all these records

Deletion Anomaly

Department information depends with employee

↓

WHAT IF?

we deleted the only emp. of a particular department.

↓

[we loss the department]

(*) Like emp. 10 is the only worker gives dep.

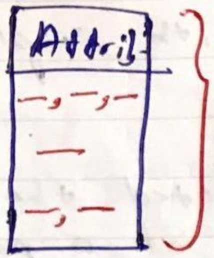
Summary

Insert: can't add dep without emp. (Need) (depend)

Update: have to update all (All)

Delete: dep will be deleted if delete the only/last worker. (Depend)

Repeating Groups

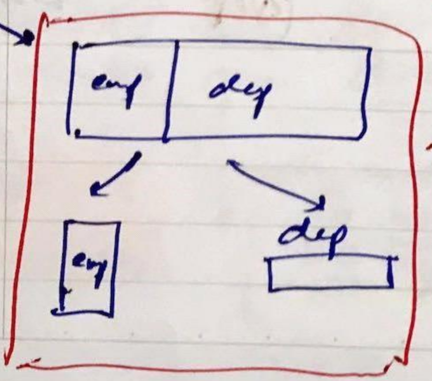


✗ can't be like this since all attributes have to be atomic (no multivalued)

Schema Refinement

Redundancy → can be refined → by decomposing relation → into smaller relations.

> contain same information
↓
With no redundancy

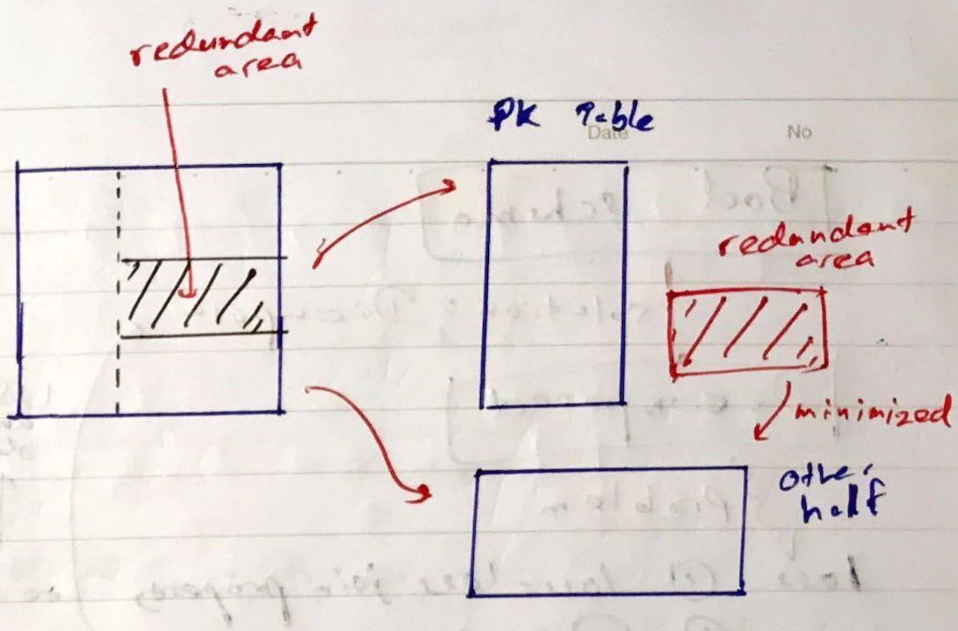


emp

10	–
51	–
40	–
45	–
46	–

dep

10	–	–
20	–	–
30	–	–
40	–	–



Problems related to decomposition.
← have to preserve.

① Loss-less join property and
② Dependancy preserving property

Normal forms → ~~have been~~ to preserve those properties

When we are decomposing

⇒ Data Loss-less (nothing lost) ①

⇒ Relationships didn't loss ②

↑ With Normal form we can verify that we didn't loss any.

if we are using NF ② will not be lost

Bad schema

↓ solution: Decompose

Decomposed

↓ Problem

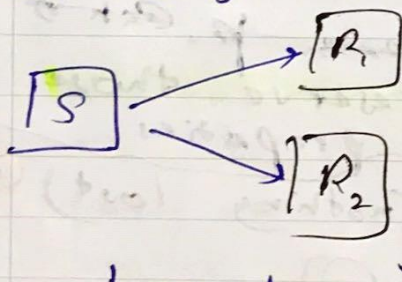
loss ① loss-less join property
② Dependency preserving property

USE NF and do the decomposition

↑ to store

(*) ① If you use NF to decompose
② Problems won't be happened.

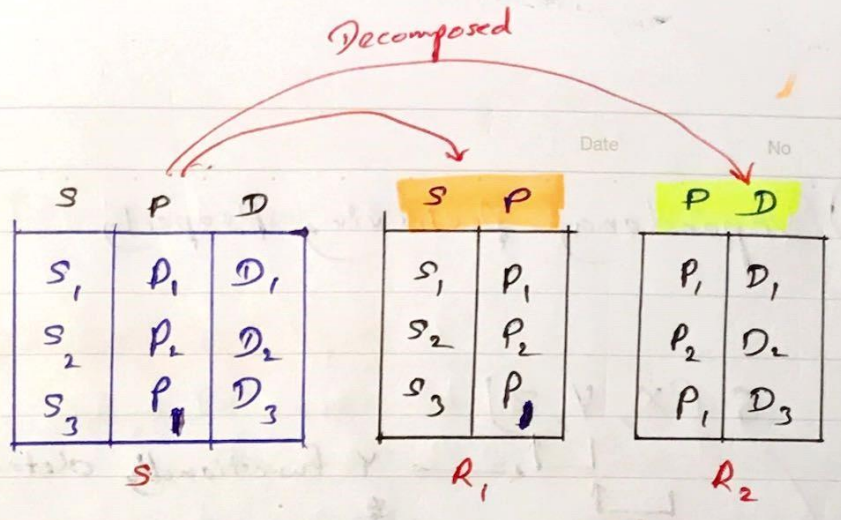
Loss-less join property



Decomposed tables

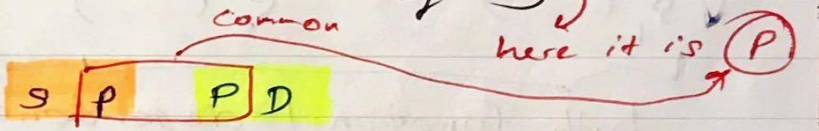
Loss-less means:

if we combine R_1 and R_2 together → original (S) table should be reformed,

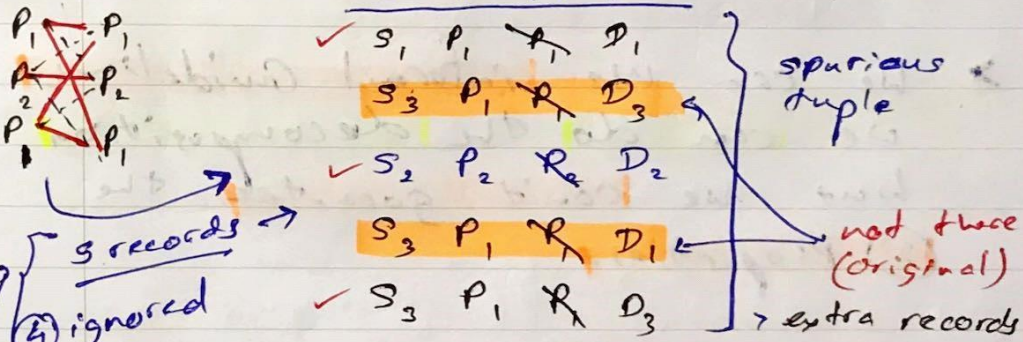


$R_1 \bowtie R_2$ (R_1 join with R_2)

When we are joining consider the equality



$R_1 \bowtie R_2$



5 records → ④ ignored

9 combinations were there.

$(3 \times 3) = (3+3+3)$

∴ This decomposition is lossy. (Not loss-less)

(*) 1) Dependancy preserving Property

S (X, Y, Z)

Y functionally determines Z

X functionally determines Y

Decomposed

S_1 ( ) S_2 ( )

both functional dependancies should exist wither the decomposed tables.

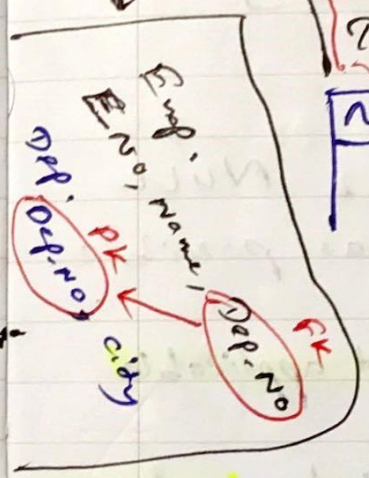
➤ We can use informal Guidelines we can do the decomposition but we can't garantee the 2) Properties.

Informal Guidelines

(*) Guiddeline 1:

Bro, If this table is about employees, wtf are you doing by adding fucking department information ~~into~~ in to that?

make it like this



The very first example. Wtf?

No	job	dep No	dep name	dep city

WTF?

if you wanna assign particular employee in to particular department

Use ~~pri~~ foreign keys

and don't do this!

No need of the decomposition anymore then, but this is informal.

Guideline (2): No redundancy

make a schema with no
- → insertion
- → update
- → deletion

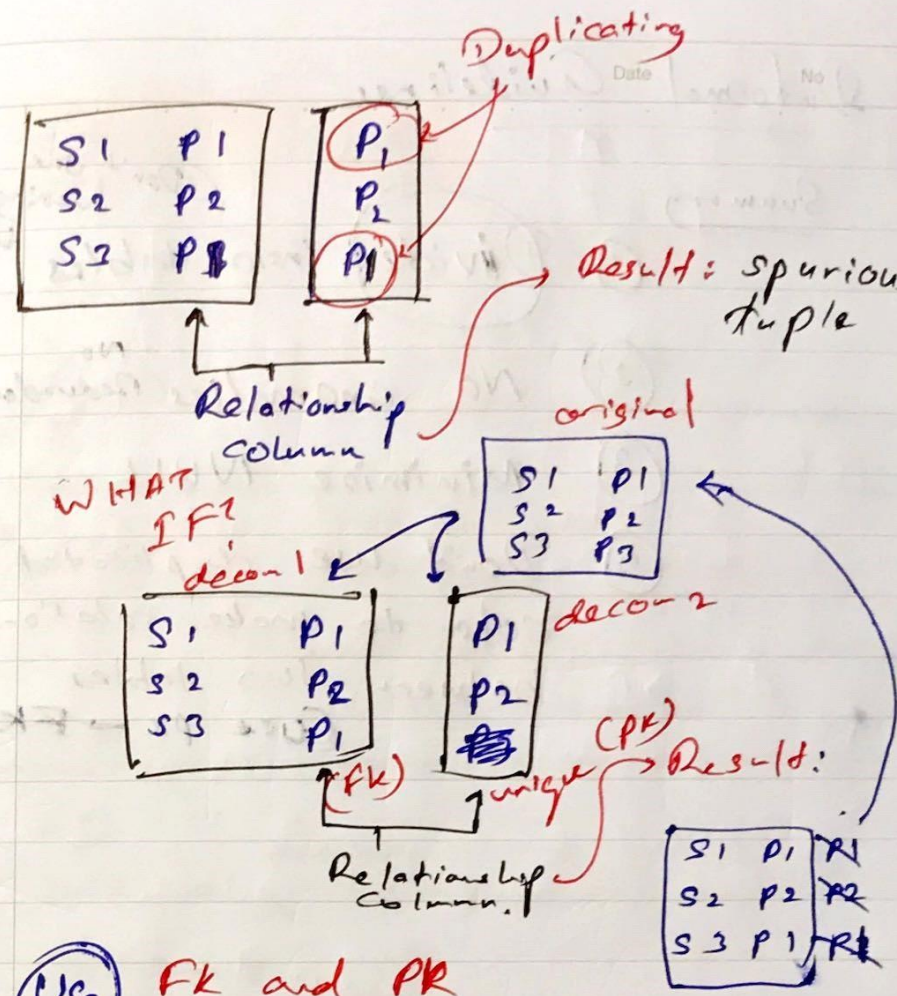
} anomalies

again we (Fk) and we other table

Guideline (3): Minimize NULL values as possible.

Reasons for NULLs /
1. Attribute not applicable or invalid
2. Attribute value unknown
3. Exist, not available

Guideline (4): If we used duplicating key ~~(not PK)~~ to make the relationship, spurious duples will be generated when we combine.



Usa: Fk and PK or some unique feild of one table to resolve this problem

Informal

Informal Guidelines

Summary

1. Divided into tables (Don't one fucking table)
2. No anomalies (No Redundency)
3. Minimize NULL
4. Don't use duplicated cols to make relation-ship between two tables (use pk → FK)

Formal Process

→ is Normalization

↓

and it based on functional dependencies (FD)

Functional Dependency

FD → used to measure Goodness of relationship

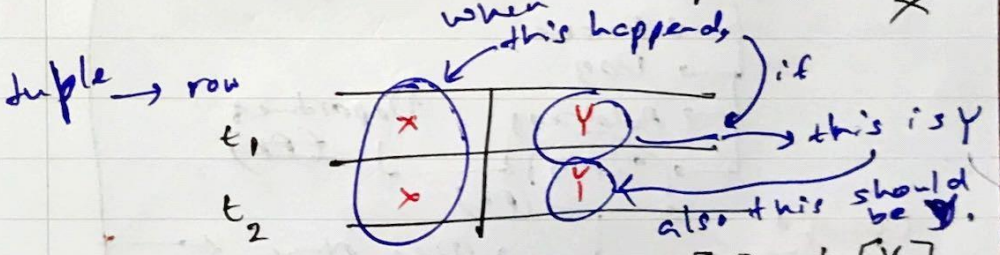
FD → used (also keys) to define NF for relation.

FD is a constraint between two sets of attributes.

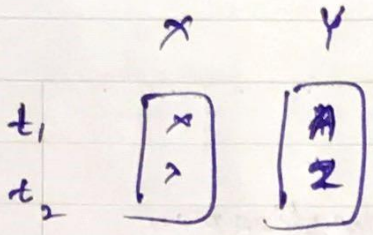
$X \longrightarrow Y$

> X functionally defines Y

> Y functionally dependent on X



if $t_1[X] = t_2[X]$ then → $t_1[Y] = t_2[Y]$



if, $t_1[X] = t_2[X]$

$\downarrow$

$(A = Z)$

it means,

$t_1[Y] = t_2[Y]$

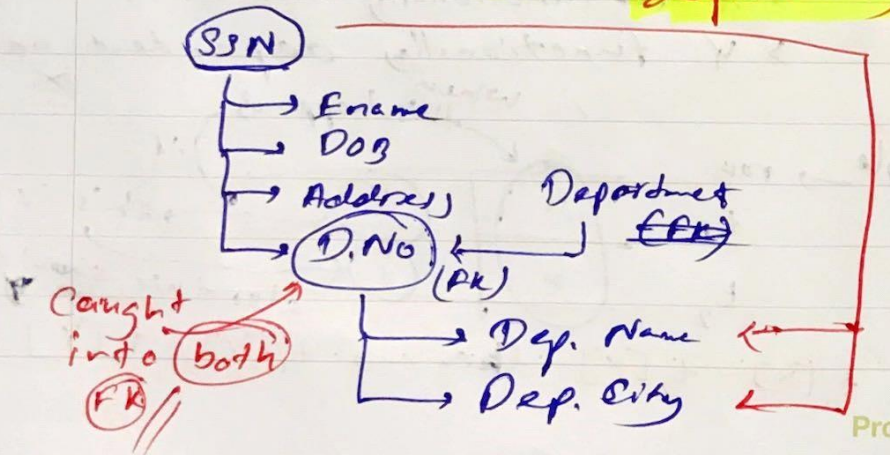
(FD) Describe the relationship between attributes in a relation.

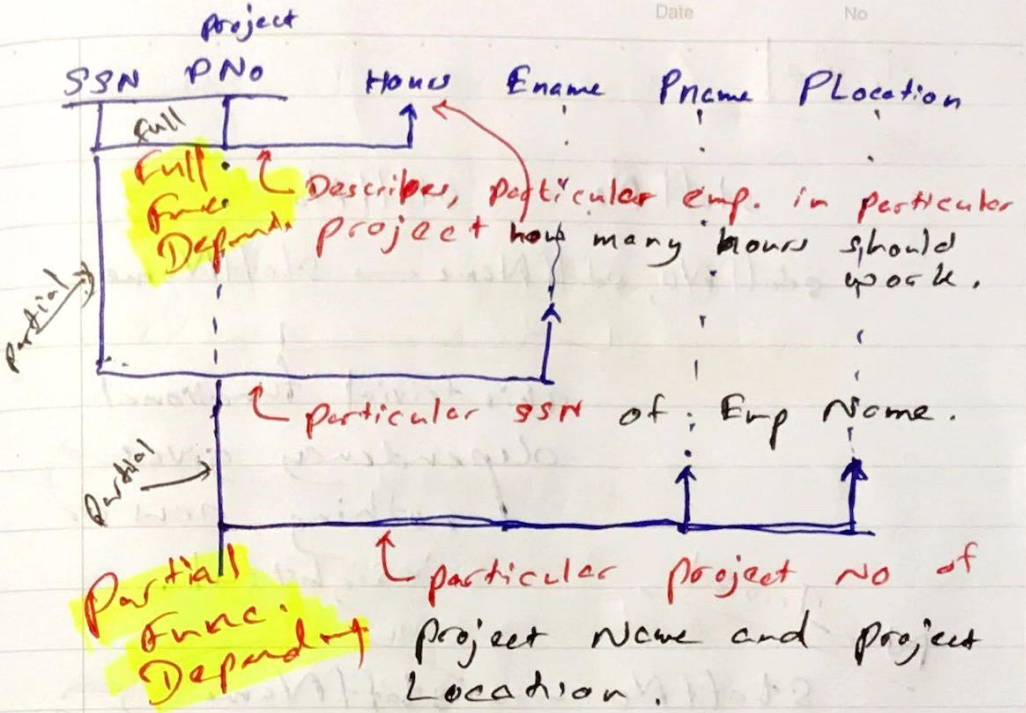
A → B

↳ dependent

(*) A and B may each consists of one or more attributes.

Transitive functional dependency





if, ~~PNo = 12~~

	PNo	PName	PLocation
t_1	12	x	y
t_2	12	x	y

also, if ~~PNo~~ PNo = 12 Those should be ~~x~~ x ~~y~~ y

$\because$ PNo → (Pname, Plocation)

$\therefore$ $t_1[PNo] = t_2[PNo]$

then $t_1[PName] = t_2[PName]$ and $t_1[PLocation] = t_2[PLocation]$ should.

Trivial

staffNo. ⟶ staffNo

staffNo, staffName ⟶ staffName

This trivial functional dependency gives nothing new or insightful.

Non-Trivial

staffNo ⟶ staffName (new information)

This non-trivial functional dependency gives new.

Functional dependency

→ interface rules (IR)

IR1 Reflexive

if, Y subset of X then X ⟶ Y (trivial)

~~IR 2~~ ~~Augmentation~~

~~if, X ⟶ Y then~~

~~XZ ⟶ YZ~~

~~sid ⟶ name~~

~~then, using reflexive law~~

~~sid ⟶ sid~~

~~sid ⟶ name~~

~~sid ⟶ sid, name~~

mostly trivial

IR2 Augmentation rule

eid → name

↳ then,

eid, address → name, address

IR3 Transive rule

eid → name
name → dept.

eid → dept

Additionally,

① (eid) → (name, dept.)

Decomposition

eid → name.
eid → dept.

② ~~Use~~ Union

eid → name eid → dept.

eid → name, dept

③ ~~eid → name~~ ~~name, dept →~~

Pseudotransitivity

③ eid → name

city, name → distance

city, eid → distance

Closure

- Student = S
  Course = C
  Instructor = I

(SID) → (S_name)

(CID) → (C_name)

(CID) → (IID)

(IID) → (I_name)

→ IR1 reflexive
  > Sid → Sid (Trivial)
  > Cid → Cid (Trivial)
  > IId → IId (Trivial)

IR2 Augmentation

IR3 Transitive by applying them,

closure can be found.

Here,

Sid → (S_name, C_name, ...)

closure. all the attributed,

Unnormalized

Normalization

step ① : check multi valued attrib and correct
(check if each and every is atomic)

↓

1st Normal form

↓

step ② : Partial dependencies can't be held.

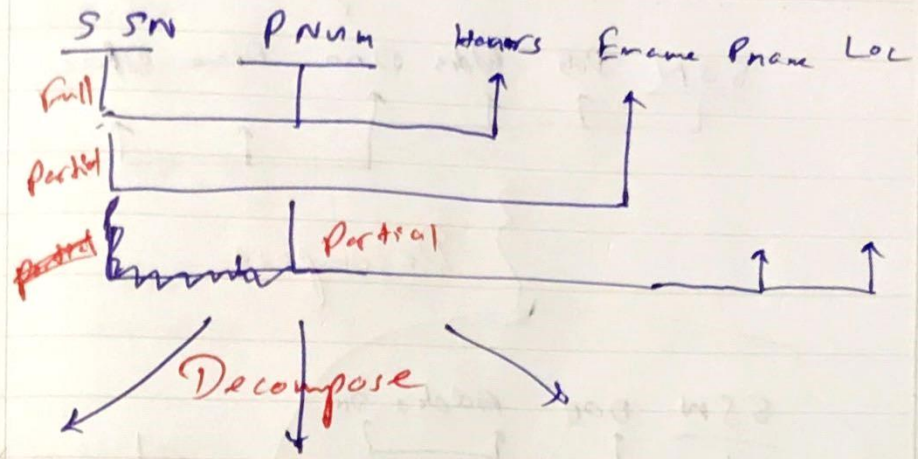
Full dependency, assume remove one part of key

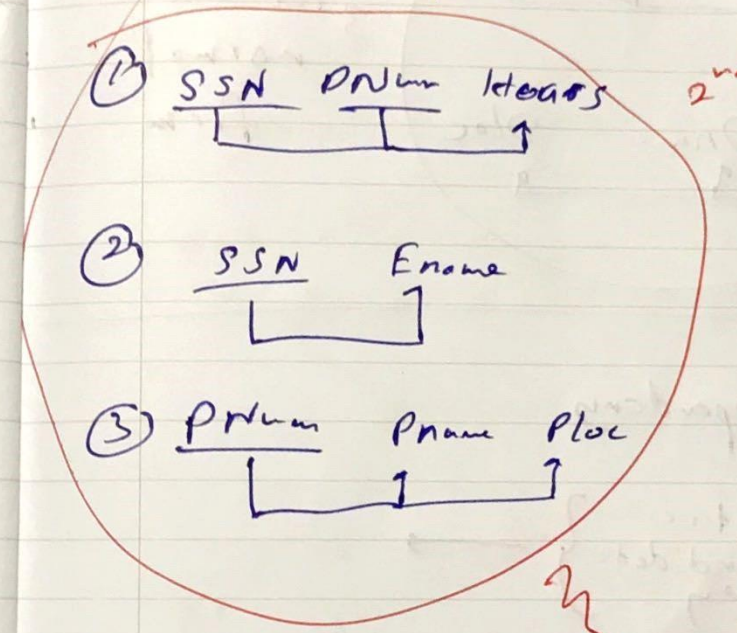
↓

now this is a partial dependency

↓

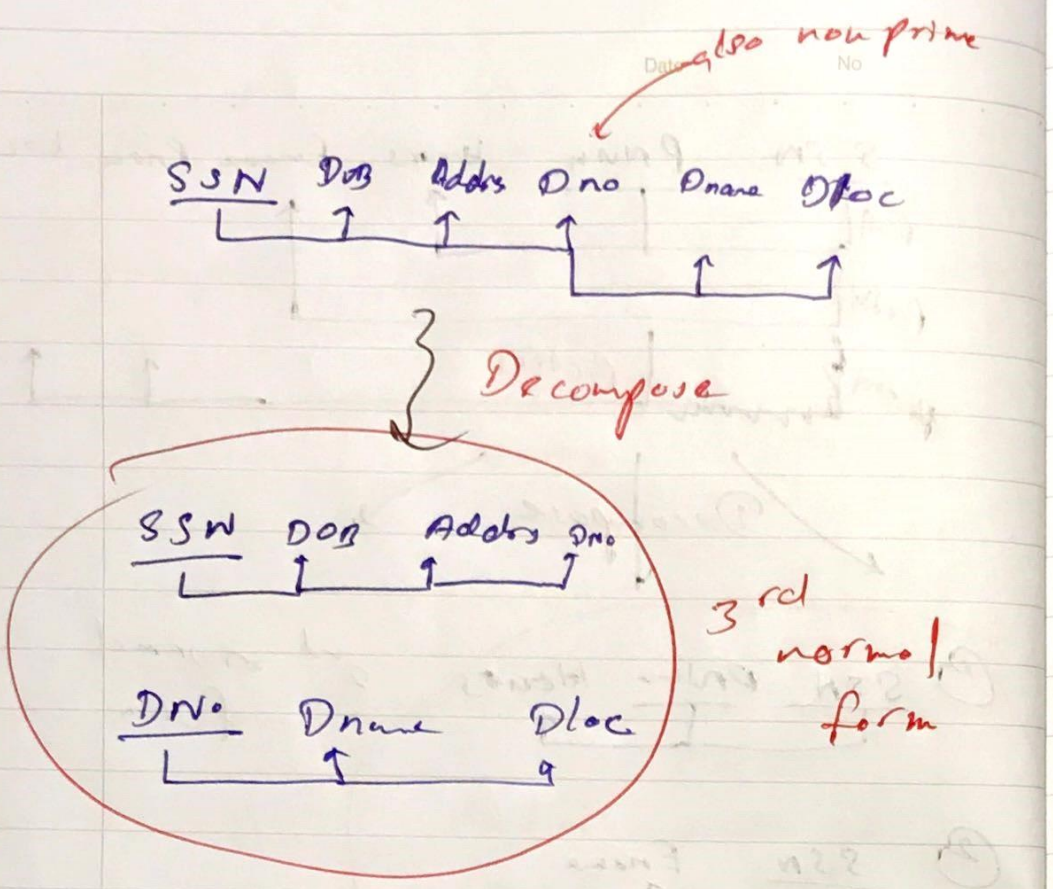
this partial could be there.





2nd normal form

step ③ : remove Transitive dependency

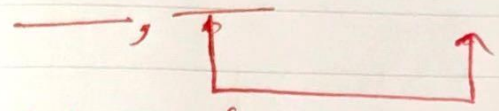


Full dependancy

entire candidate key →

key attribute a functionality determined by a full key

Partial dependancy



Subset of candidate key

Transitive

Prime → non-prime

non-prime → non-prime