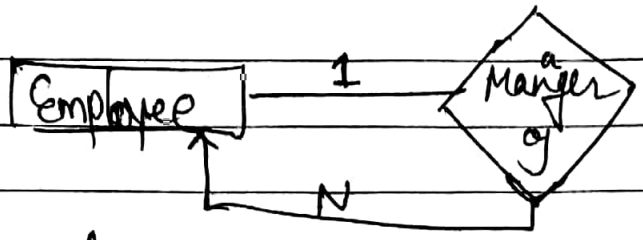
* Unary rel^n to schema example

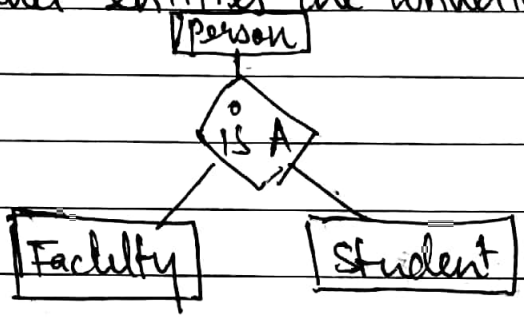


Emple (Emp.No, Name, ..., Manager ID)

* Generalization (Bott^m up)

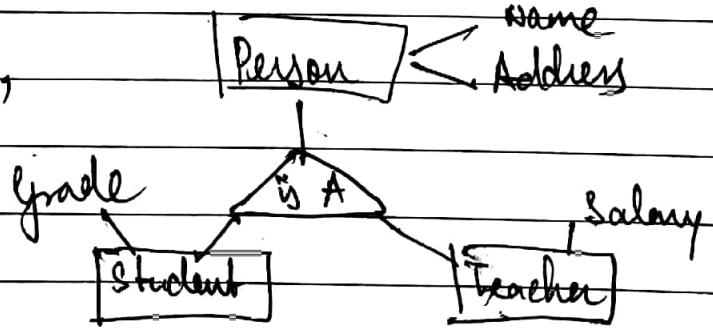
Two lower level entities are converted to a higher level entity.

Ex:-



Ex:- Student (Name, addres, grade)
Faculty (" " Salary)

After gen^n,



* Specialization (Top dw^n)

* Aggrg^n - In agg^n, rel^n b/w two entities is treated as a single entity. In aggr^n, relt^h with its corresponding entities is aggregated into a higher level entity.

Normalizn

redundant

or reduce

→ Technique to remove redundancy from the table.

Three types of anomalies
1) Insertion anomaly
2) Deletion "
3) Updation "

1 NF : Table should only contain atomic attributes.

Closure Method

R(ABCD) , FD { A → B, B → C, C → D }

$A^+ = ABCD$ ∴ A → candidate key.

$B^+ = BCD$

$C^+ = CD$

$D^+ = D$

$AB^+ = ABCD$

But here AB ≠ C.K. , it is super key.

Prime attributes:- Used for making C.K.

Ex:- R(ABCDE)

FD = { A→B, BC→D, E→C, D→A }

∴ $E^+ = EC$

In FD, in RHS there are A, B, C, & D. ⇒ For CK determination E should be there on LHS only then E will appear on RHS.

∴ $(AE)^+ = AEBCD$. CK = { AE, BE, DE, }

$(BE)^+ = BECDA$

$(CE)^+ = CE$

$(DE)^+ = DEACB$

Functional Dependency:- Method of describing relⁿ b/w attributes.

X → Y (X determines Y)

Trivial Fⁿ Dependency.

X → Y & Y is a subset of X.

$X \cap Y \neq \phi$

Scanned with CamScanner

Non trivial fn dep^n

$X \to Y$, $Y \not\subseteq X$

$X \cap Y = \phi$

Properties: 1) If $X \to Y \Rightarrow XZ \to YZ$

2) If $X \to Y$ & $Y \to Z \Rightarrow X \to Z$

3) $X \to Y$ & $X \to Z \Rightarrow X \to YZ$

3) $X \to YZ$ then $X \to Y$ & $X \to Z$

Note: $XY \to Z$ $X \not\to Z$, $Y \not\to Z$

4) $X \to Y$ and $WY \to Z \Rightarrow XW \to Z$

$X \to Y$ and $Z \to W \Rightarrow XZ \to WY$

2NF: 1^st NF + no partial dependency.

R(ABCDEF)

Ex: FD:- { $C \to F$, $E \to A$, $EC \to D$, $A \to B$ }

CK:

~~$(EC)^+ = EFAB$~~

$(CE)^+ = CEFDAB$

$C^+ = CF$, $E^+ = EAB$

$\Rightarrow CK = \{CE\}$

ABDF, (CE)

⇒ Prime attributes. CE

Non " " : A, B, D, F

∴ C → F

↳ Partial dependency ⇒ Not 2NF.

↓

Part of a C.K. determines a non prime attribute.

For partial dependency,

LHS should be a proper subset of CK ⚠

RHS " " " non prime attr.

3NF → 2NF + No transitive dependency.

↓

⚠ i.e. a non prime attr^n cannot determine any non prime attr.

Ex. R(ABCD)

AB → C, C → D

CK = AB

↳ Non prime → Non prime

⇒ Not in 3NF.

Eg: R(ABCD)

AB → CD, D → A

B: B BD: BDA.

CK = {AB, BD}

$(AB)^+$ = ABCD

$(BC)^+$ = BC

$(BD)^+$ = BDAC

P.A = A, B, D

N.P.A = C

~~A → A is a partial dependency.~~

To check valid FD for 3NF.

· LHS should be a CK or SK [or] RHS should be a prime attr.

∴ AB → CD (Valid)

D → A (")

∵ For 3NF, if NPA determines NPA, only then it's a problem else not.

BCNF

↓

In all the F.D's given, LHS should be a CK. or a SK.

* Lossless and Lossy decompⁿ

Common attribute should be C.K. or SK of either R_1 or R_2 or both.

Ex:

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

$R_1 \bowtie R_2$

Here commⁿ attr should be A bcoz its CK & not B & C because they contain duplicate values.

$\therefore R_1(AB) \bowtie R_2(AC)$ } Lossless join decompⁿ.

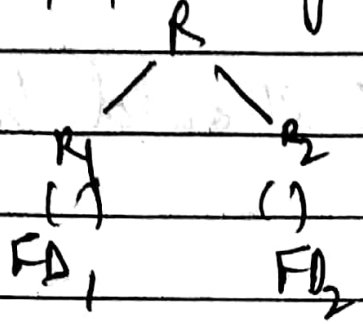
R_1

A	B
1	2
2	2
3	3

R_2

A	C
1	1
2	2
3	2

Dependency preserving decomp^n



$\therefore FD_1 \cup FD_2 = FD$

Ex:- $R(ABCD)$, $FD = \{A \to B, B \to C, C \to D, D \to B\}$

R is decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(BD)$

$R_1(AB)$

$A \to A$ } Trivial
$B \to B$ }
$A \to B$
$B \to A$

$\therefore$
$A \to B$ & $B \to A$

$B^+ = BCD$
$\Rightarrow B \not\to A$
$\therefore R_1(AB)$

$A \to B$

$R_2(BC)$

$B \to C$
$C \to B$
$C^+ = DB$
$\Rightarrow C \to B$
$\therefore R_2(BC)$
$B \to C, C \to B$

$R_3(BD)$

$B \to D$ & $D \to B$
$B^+ = BCD$
$\Rightarrow B \to D$
$\therefore B \to D$ & $D \to B$

$\therefore FD_1 = A \to B$
$FD_2 = B \to C, C \to B$
$FD_3 = B \to D$ & $D \to B$

$C \to B$ & $B \to D$ $\Rightarrow C \to D$

$FD_1 \cup FD_2 \cup FD_3 = \{A \to B, B \to C, D \to B, C \to D\}$

Ex: $R(ABCD)$, $FD = \{AB \to CD, D \to A\}$

$R_1(AD)$ | $R_2(BCD)$

$A \to D$, $D \to A$

$A^+ = A$

$\Rightarrow A \not\to D$

$B \to CD$ ✗

$C \to BD$ ✗

$D \to BC$ ✗

✗ $BC \to D$, $CD \to B$, $BD \to C$

~~BCD~~ $\{BCD \to B, BCD \to C, BCD \to D\}$ Trivial

$\therefore B^+ = B$, $C^+ = C$, $D^+ = DA$

$BC^+ = BC$, $BD^+ = BDAC \therefore BD \to C$ ✓

$CD^+ = CDA$

$FD_1 = \{D \to A\} \Rightarrow FD_2 = BD \to C$

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

~~$BD^+ = BDAC$~~

~~$\Rightarrow BD \to AB$~~ CD

From here we need to check if $AB \to CD$

$\therefore AB^+ = AB \Rightarrow FD_1 \cup FD_2$ doesn't presume $AB \to CD$.

Convert a rel^n from 1NF to 2NF.

Eg: $R(ABCD)$, $F = \{A \to B, B \to C\}$

~~$CK = \{A\}$~~ $CK = AD$

$\therefore PA = A, D$, $NPA = B, C$

(i) $(A \to B) \to$ Violates 2NF.

$A \to B$ ∵ (Partial dependency)

$B \to C$ ∵ (Not a " ")

For decomp^n into tables, the common attr. should be a super key of ~~atts~~ at least one of the ~~tables~~ subrelation.

$\therefore A \to B$ is causing problem

$\Rightarrow A^+ = ABC$

$\therefore R_1 = \langle ABC \rangle$ & $R_2 \to$ remaining attributes i.e. D

$R_1 = \langle ABC \rangle$, $R_2 = \langle AD \rangle$

Here common attribute should be A because for R_1, $A^+ = ABC$ $\Rightarrow$ its a SK.

$\therefore R_1 \langle ABC \rangle$, $R_2 = \langle AD \rangle$

If $R_2 = BD$ or CD then its lossless because $B^+ = BC$ & $C^+ = C \to$ Not SK of R_1 or R_2

$R_1(ABC)$ | $R_2(AD)$ → BCNF (only two attributes)

$A \to D$, $D \to A$

$F_1 = \{A \to BC, B \to C\}$ | $A^+ = AB$ $A^+ = A$, $D^+ = D$

→ $F_2 = \{\}$

R_1, $F_1 = (A \to BC, B \to C)$

$CK = A$ → Not a partial dependency (PD)

→ R_1 is in 2NF

∵ if CK is just a single attr. then no chances of PD bcoz no proper subset of CK.

$F_1 \cup F_2 = (A \to BC, B \to C)$

& $F_2 (A \to B, B \to C)$ ∴

∴ $A^+ = ABC \Rightarrow A \to B$

$B \to C$

∴ $F = F_1 \cup F_2$ ⇒ Dependency preserving decomp^n

Ex. R(ABCD)

FD = { A → B, C → D }

CK = AC (B D)

$C_K = (AC)^+ = ACBD$

PA = A C, NPA = B, D

∴ A → B & C → D are PD's

⇒ A → B & C → D both are creating problems.

$A^+ = AB$ $C^+ = CD$

$R_1 = \langle AB \rangle$, $R_2 = \langle CD \rangle$

For common attribute, we can choose A or C because it's SK of R_1 & R_2 respectively.

$R_1 = \langle AB \rangle$, $R_2 = \langle ACD \rangle$

↓

BCNF ∵ two only attr

$F_2 = \{ A \to CD, C \to D, AC \to D \}$

$A^+ = ACD$

$C^+ = CD$

$D^+ = D$

$AC^+ = ACD$

$CD^+ = CD$

$AD^+ = AD$

$F_1 = A \to B$

For the remaining attributes.

$A^+ = AB$ $\quad$ $C^+ = CD$

$R_1 \langle AB \rangle$ $\quad$ $R_2 \langle CD \rangle$ $\quad$ R_3

But no remaining attributes.

If we join $R_1 \bowtie R_3$, we can use A as common. For $R_2 \bowtie R_3$ join, C as common.

$\Rightarrow R_3(AC)$

$R_1(AB)$	$R_2(CD)$	$R_3(AC)$
CK = A	CK = C	$A \to C$, $C \to A$
$F_1 = A \to B$	$F_2 = C \to D$	$F_3 = ?$ $\therefore F_3 = \{\}$

$F_1 \cup F_2 = F$

R_1, R_2, R_3 in BCNF & only two attributes.

Note: FD's which create a problem, create a separate table for them.

Recurring

R(A B C D E)

Acc to NJ Rule

$A \rightarrow BCDE$

$C \rightarrow BDE$

$D \rightarrow BE$

$C^+ = CBDE$

$C^+ = CBDE$

(*)

ABCDE

$AC \rightarrow D$

$D \rightarrow E$

$E \rightarrow A$

$DE \rightarrow A$, BC

~~$(BC)^+ = BC$~~

$(BCA)^+ = BCADE$

$(BCD)^+ = BCDEA$

$(BCE)^+ = BCEAD$

2NF

Partial dependency

pr → Non prime attr

ABCDE

Non prime attr

Convert a relⁿ from 2NF to 3NF

Eg. $R(ABCD)$ $F_D = \{A \to B, B \to C, C \to D\}$

$CK = A$ ∴ $B \to C, C \to D$ (Violating 3NF)

transitive dependency.

∴ $B^+ = BCD$	$C^+ = CD$	
$R_1 \langle BCD \rangle$	$R_2 \langle CD \rangle$	$R_3 \langle BA \rangle$

Common attribute should be B ~~$R_3 \langle ABC \rangle$~~ $B^+ = BCD$

and not both B & C because &so B determines R_1 & R_2 both.

∴ $R_1(BCD)$	$R_2(CD)$ BCNF	$R_3(AB)$ BCNF
~~$B \to C, C \to D$~~	$C \to D$	$A \to B$
~~$BC \to D, BD \to C$~~,	F_2	F_3
~~$B \to D$~~		
F_1		
$BC^+ = BC$ $B \to CD$,		
$CD \to B$ $C \to BD$		
$CD^+ = CD$		

∴ ~~$F_1 \cup F_2 \cup F_3 =$~~

this accounts for $B \to C$, $B \to D$ etc.

∴ No need to write

For R_1 | CK = B

$F_1 = \{ B \to CD, C \to D \}$

Still a transitive dependency

$\Rightarrow$ R_1 still in 2NF

$\Rightarrow$ Now decompose only R_1. $R_1(BCD)$

$\therefore C \to D$ is a problem.

$C^+ = CD$ (From F_1)

$R_{11} < CD >$ $R_{12} < BC >$

Common attr is C.

$R_{11} < CD >$: $R_{12} < BC >$

$\therefore R_{11}$ & R_{12} $\to$ BCNF.

$R_{11}(CD)$	$R_{12}(BC)$
$F_{11} = \{ C \to D \}$	$F_{12} = \{ B \to C \}$

$\therefore F_{11} \cup F_{12} \{ C \to D, B \to C \}$

$B^+ = BCD \Rightarrow B \to CD$ & $C \to D$

$\therefore$ Dependency preserved.

~~ii $R_4(AB)$, R_5 {~~

Final decomp^n → $\underbrace{R_1(AB), R_2(BC), R_3(CD)}_{BCNF}$

* Cannonical Cover / Minimal Cover / Irreducible set of FDs

For a f^{n}al dependency F, F' is a cannonical cover if F' doesn't have (i) Extraneous Attr (ii) redundant FD.

Steps: (i) Splitting rule so that RHS has only single attribute.

Eg. $A \to BC \Rightarrow A \to B$ & $A \to C$

But $AB \to C \to$ ~~(Reflexivity)~~ $A \to C$ & $B \to C$

(ii) Remove extraneous attribute (i.e. we'll remove this from RHS only).

(iii) Remove redundant FD.

Example of removal of extraneous attribute

If $AB \to C$ & $A \to C$ then we can remove B because it's an extraneous attribute.

Ex:- If $AB \to C$ & $A \to B$ still B is extraneous in $AB \to C$ ∵ using A we can find B & then using AB, we can find C.
∴ $A \to C$ & $A \to B$.

Ex. $AB \to C$ & $B \to C \Rightarrow A \to C$ & $B \to C$
∴ B is extraneous

$AB \to C$ & $B \to A \Rightarrow$ A is extraneous
∴ $B \to C$ & $B \to A$.

4NF

Cond's for 4NF → (i) R is already in 3NF or BCNF.
(ii) if it contains no MVD.

MVD:- It is the dependency where one attribute value is potentially a multivalued fact about another.
F.D. ($\alpha \to \beta$) says we can't have diff^n values of β for same value of α.
In MVD, for same value of α we can have diff^n value of β. i.e. MVD:- $\alpha \to\to \beta$

For MVD, imp point.
(i) There must be atleast 3 attributes
(ii) Attri. must " independent of each other.

Ex.

Person (P)	Mobile (M)	Food_like (F)
P_1	M_1	F_1
	M_2	F_2
P_2	M_3	F_3

→

	α	β	
	Person(P)	Mobile(M)	Fwd_like(F)	
$t_4 \rightarrow$	P_1	M_1	F_1	$(P \twoheadrightarrow F)$
$t_2 \rightarrow$	P_1	M_1	F_2	$(P \twoheadrightarrow M)$
$t_3 \rightarrow$	P_1	M_2	F_1	
$t_4 \rightarrow$	P_1	M_2	F_2	
	P_2	M_3	F_3	

MVD:- $\alpha \twoheadrightarrow \beta$. If any legal rel^n $r(R)$ for all pairs of tuples t_1 & t_2 in r such that $t_1[\alpha] = t_2[\alpha]$ then there exists tuples t_3 and t_4 in r such that

1) $t_3[\alpha] = t_4[\alpha] = t_1[\alpha] = t_2[\alpha]$
2) $t_3[\beta] = t_1[\beta]$
3) $t_4[\beta] = t_2[\beta]$

If these satisfy then $\alpha \twoheadrightarrow \beta$

* MVD occurs if two or more independent rel^{n}s are kept in a single rel^n. Consider two rel^{n}s

R_1 (SID, Sname), R_2 (CID, Cname)

Both rel^{n}s are independent of each other.

R_1

SID	Sname
S_1	A
S_2	B

R_2

CID	Cname
C_1	C
C_2	B

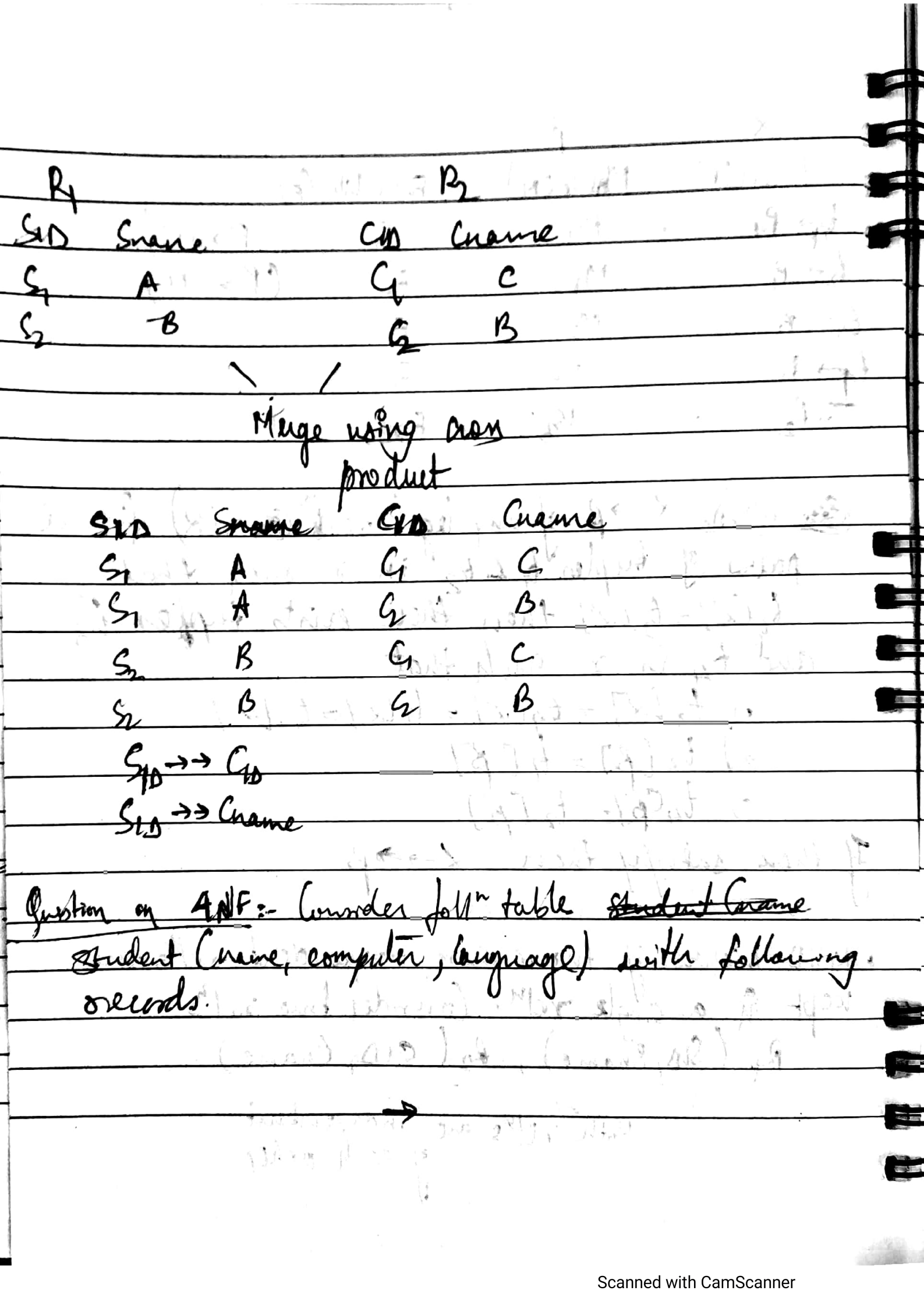
Merge using cross product

SID	Sname	CID	Cname
S_1	A	C_1	C
S_1	A	C_2	B
S_2	B	C_1	C
S_2	B	C_2	B

$S_{ID} \rightarrow\rightarrow C_{ID}$

$S_{ID} \rightarrow\rightarrow$ Cname

Question on 4NF:- Consider following table ~~Student Cname~~ Student (Cname, computer, language) with following records.



Name	Computer	Language
Aman	Windows / Apple	English, Hindi
Mohan	Linux	English, Spanish

Normalize the table. Is it in 4NF? If no, decompose it into 4NF.

Name	Computer	Language
Aman	Windows	English
"	"	Hindi
"	Apple	"
"	"	English
Mohan	Linux	"
"	"	Spanish

~~Name →→ Computer, Name →→ Language~~

Name →→ Computer , Name →→ Language] MVD.

→ R_1 → R_2

To remove MVD, make separate rel^{n}s for those which have MVD i.e. →

(11)

(13) $x = t^2$, $2t\,dt = dx$, $\int^{?}$

R_1

Name	Computer
Aman	Windows
Aman	Apple
Mohan	Linux

There no MVD because only two attributes

Key of R_1 : (Name, computer)

R_2

Name	Language

R_2 not MVD because 2 attributes

Key of R_2 (name, language)

5NF (Projected join normal form)

(i) R is in 4NF (No MVD in R)

(ii) It cannot be further non loss decomposed

Join dependency :- Let 'R' be a rel^n schema and $R_1, R_2, \dots R_n$ be the decomp^n of R then R is said to satisfy the join dependency $*(R_1, R_2, \dots, R_n)$ iff

$\Pi_{R_1}(R) \bowtie \Pi_{R_2}(R) \cdots \bowtie \Pi_{R_n}(R) = R$

(Order of join doesn't matter)

(OR)

iff every legal instance $r(R)$ is equal to join of its projections on $R_1, R_2 \ldots R_n$.

Ex of join dependency

Agent	Company	Product
Aman	C1	Pen drive
Aman	C1	Mic
"	C2	Speaker
Mohan	C1	Speaker

→ This table has redundancy.

$\bowtie$ R_1 & R_2

(Agent | Company) ⋈ (Agent | Product)

If $R_1 \bowtie R_2 = R$ then J.D holds.

(or)

Let R_3 = (Company | Product) & if $R_1 \bowtie R_2 \bowtie R_3 = R$

$\Rightarrow$ There's J.D.

If there's JD i.e. R can further be decomposed & hence not in 5NF.

$R_1 \bowtie R_2 \neq R$ but $R_1 \bowtie R_2 \bowtie R_3 \neq R$

& Find JD's in a table & if it exists then not in 5NF.

Ex:

Pname	Skill	Job
Aman	DBA	J1
Mohan	Tester	J2
Rohan	Programmer	J3
Sohan	Analyst	J4

R_1(Pname, Skill), R_2(Skill, Job), R_3(Pname, Job)

$R_1 \bowtie R_2 \neq R$ but $R_1 \bowtie R_2 \bowtie R_3 = R$

$\Rightarrow$ It has JD

$\therefore$ Not in 5NF

Serializability:- A schedule which is one serializable.

Ex: S

T_1	T_2
R(A)	
	R(A)
	W(A)
W(A)	

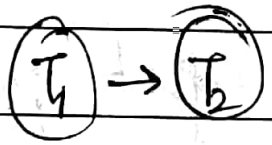
→ Parallel Schedule

→ Convert it to serial schedule

is S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)

→



To check is S is serializable or not we'll use two methods i.e. ~~abc~~ conflict & view.

Conflict Equivalent

R(A) R(A) } Non conflict pairs

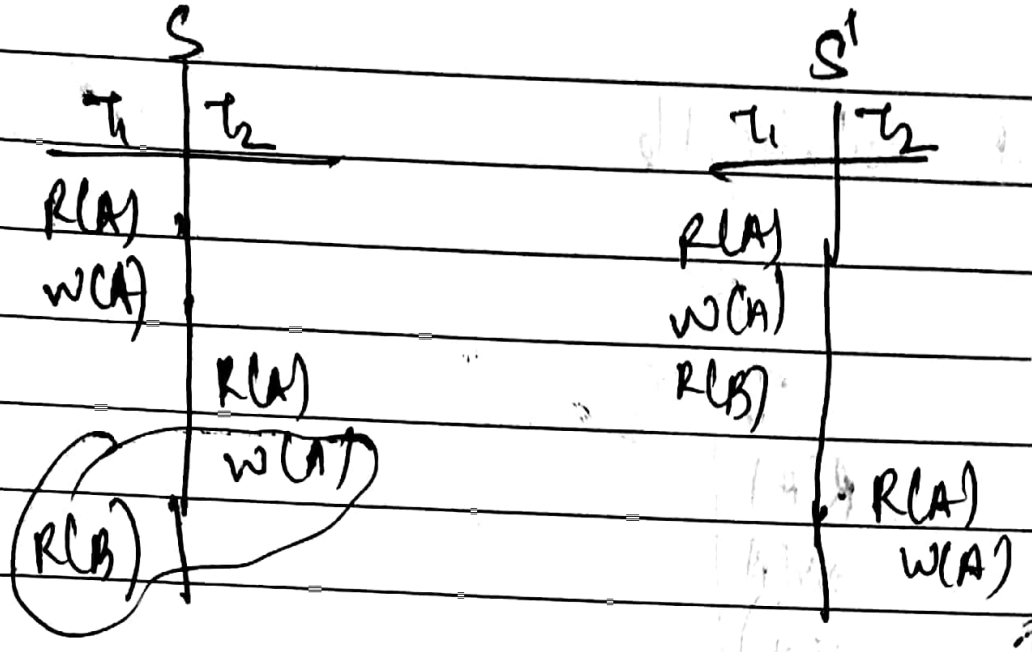
R(A) W(A) }
W(A) R(A) } Conflict pairs
W(A) W(A) }

S & S' are conflict equivalent if S can be converted to S' by some non conflicting swaps.

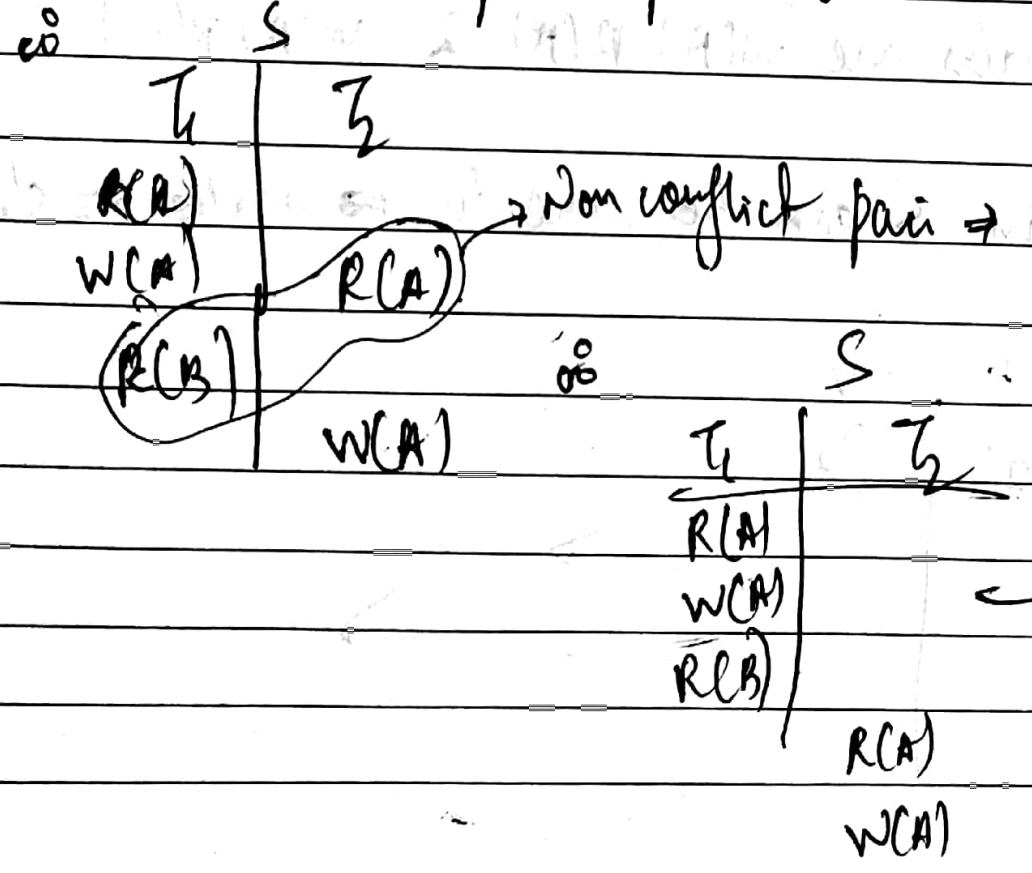
R(A) R(B) }
W(B) R(A) } Non conflict pairs:
W(A) R(B) }
W(A) W(B) }

∴ S is conflict serializable if it is conflict equivalent to a serial schedule.

Ex →



To make S conflict serializable to S', check adj^n non conflict pairs. In above case its R(B) & W(A) & then swap its post^n.



∴ S is conflict serializable

S

T_1	T_2
R(A)	
W(A)	
	R(A)
W(A)	
R(B)	

S'

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

adjⁿ pairs but none are non conflicting

⇒ S & S' are non serializable.

In S, adjⁿ pairs are W(A) R(A) & W(A) R(A).

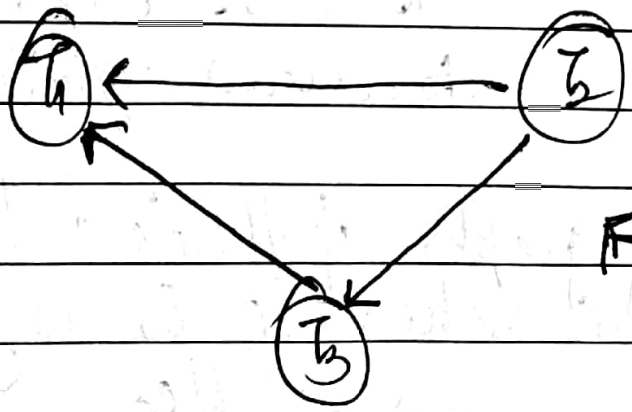
No swaps in S can be done to make S = S'.

T_1	T_2	T_3
R(x)		
		R(y)
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		

Note:
W(y) has two conflict pairs
i.e. R(y) & W(y)

To check if a schedule is conflict serializable or not.

∴ Draw precedence graph.



No loop / cycle

To draw edges, check the conflict pairs in other transactions & then draw edges.

↔ → conflict

1) For R(x), conflict pair is W(x) but W(x) is not present in T_2 nor in T_3 so cancel it.

2) Move to next i.e. R(y) ↔ W(y) → not in T2 & T1

3) Third is R(z) in T_3 R(z) ↔ W(z)
Not in T_2 but is there in T_1.

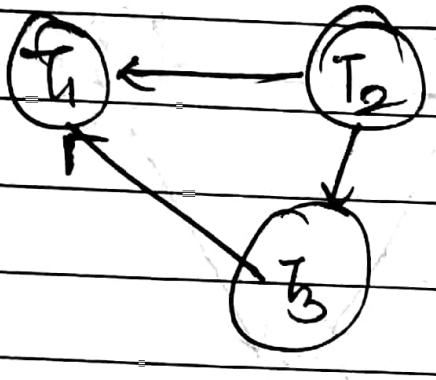
⇒ Draw a line from T_3 to T_1.

For W(z) in T_2 it has W(z) ↔ R(z) & W(z) ↔ W(z) in T_1

⇒ But $T_2 \rightarrow T_1$, there's already a line & so no need to draw again.

∴ In precedence graph if there's a loop/cycle ⇒ the schedule is conflict serializable.

∴ Since its serializable ⇒ its consistent



To find the order of schedule i.e. will it be

$T_1 \rightarrow T_2 \rightarrow T_3$

or

$T_3 \rightarrow T_1 \rightarrow T_2$

or

∴ Find a vertex whose indegree is 0.

⇒ 0 incoming edges.

Here it is T_2.

∴ After removing T_2, the graph is

Here T_3 has indegree 0.

∴ $T_2 \rightarrow T_3 \rightarrow T_1$ is the correct sequence for serial schedule.

Ex. Check if conflict serializable

S

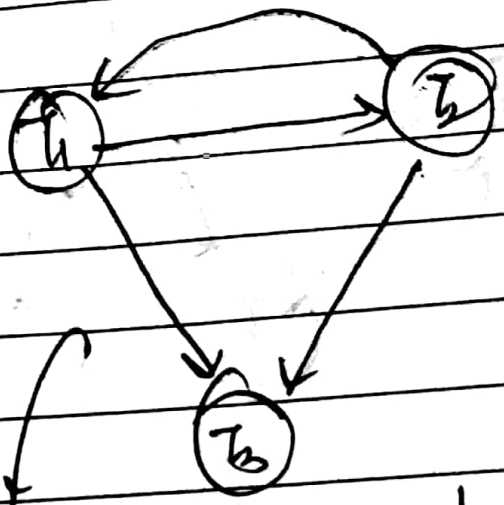
T1 T2 T3

R(A)

W(A)

W(A)

W(A)



If has a cycle ⇒ Not conflict serializable

But then we'll use second method i.e. view Serializable or not.

∴ We use view Serializable method when we find that S is not conflict serializable.

If any rel^n conflict serializable ⇒ view serializable but not vice-versa.

→

Note:- If any schedule which is not conflict serializable & is view serializable, then it must have atleast one blind write. (BW)

(V Imp)

Blind write:- If any transaction writes without any read then it's a blind write.

R(a)	W(a)
W(a)	

↓ Not blind write.

W(a) ← Blind write

∴ Conflict Serializable (C.S.) ⇒ View " (VS)

C.S
- Yes → VS
- No → (Check BW)
  - Yes → check further
  - No → Not VS.

Study View Serializability

* Irrecoverable Schedule

T_1	T_2
R(A) A=10	
A := A - 5 A=5	
W(A)	
	R(A) → A=5
	A := A - 2
	W(A) A=3
	commit ← It means that the transaction has completed.
roll back	
R(A)	
fail	

If T_1 fails here then by atomicity property, we need to rollback (i.e. undo) because atomicity says that execute either all inst's or none so we undo. If we undo then in database, A=10 again

Note:- Here T_2 will not roll back because it has already committed.

& we'
mad
is

Moreove
back
valu

Recover
order of
should
Look

Lockin
1) Share
→ Share
→ Exclu

Exclus
excl

& we've lost the changes in A which were made by T_2 & therefore the schedule is irrecoverable.

Moreover in the above eg:, only T_1 rolled back & not T_2 so in the end the value of A should be (10-2).

Recoverable Schedule :- The ones for which if the order of schedule is $T_1 \rightarrow T_2$ then it's commit order should also be $T_{C_1} \rightarrow T_{C_2}$.

→ Look its video

Locking protocols

1) Shared - Exclusive locking

→ Shared lock (S) : If the trans^n locked the data item in shared mode then it is allowed to read only.

→ ~~Exclusive lock :-~~

Exclusive lock (X) :- If trans^n locked data item in exclusive mode then allowed to write & read both.

Ex: shared lock (A)

T1	T2
S(A)	X(A) → when we need to read & write then exclusive lock
R(A)	R(A)
U(A)	W(A)
	U(A)

Unlock (A)

If only read, then shared lock.

Compatibility table

Grant \ Request	S (shared lock)	X (exclusive lock)
S	Yes	No
X	No	No

If there's S on A if there's a request for S again on A, we'll grant it but if there's a request for X, we'll not as then there can be a conflict.

Problems in S/X locking

1) May not be suff^{n} to produce any serializable schedule.

Ex:

S

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
X(B)	
R(B)	
W(B)	
U(B)	

Even after giving locks, the schedule has not become serial.

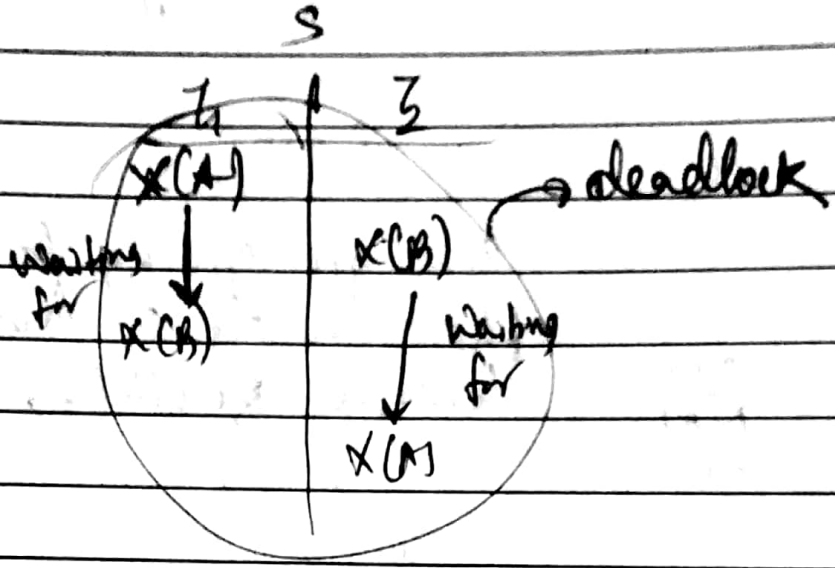
2) May not be free from irrecoverability.

S

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
	Commit
fail	

→ Still it is irrecoverable

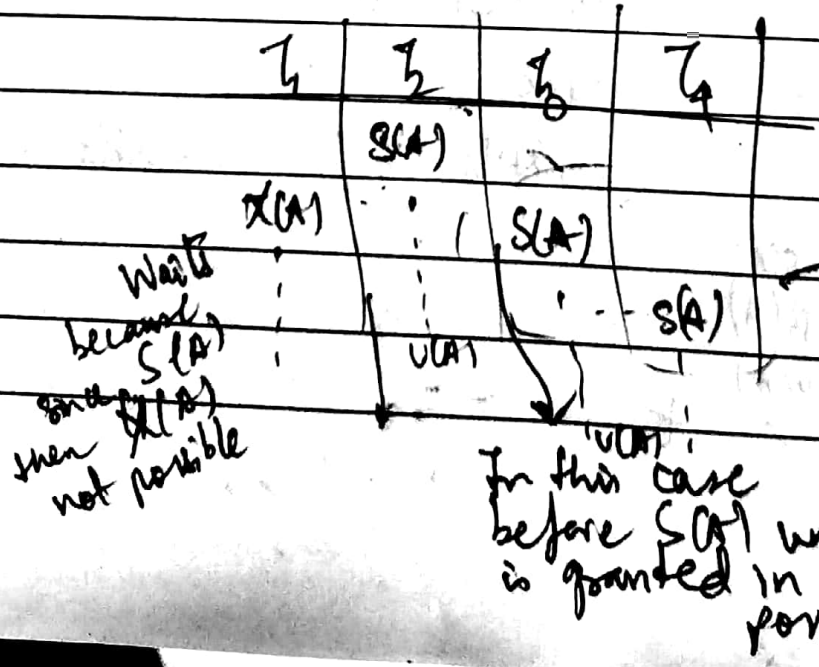
3) May not be free from deadlock



4) May not be free from starvation.

Starvⁿ vs deadlock

→ infinite waiting

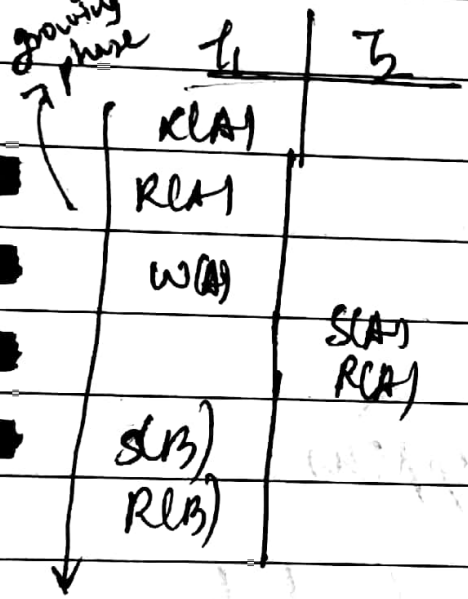


2 phase locking protocol (2PL)

→ growing phase
→ shrinking phase

** Because of this serializability is assured.

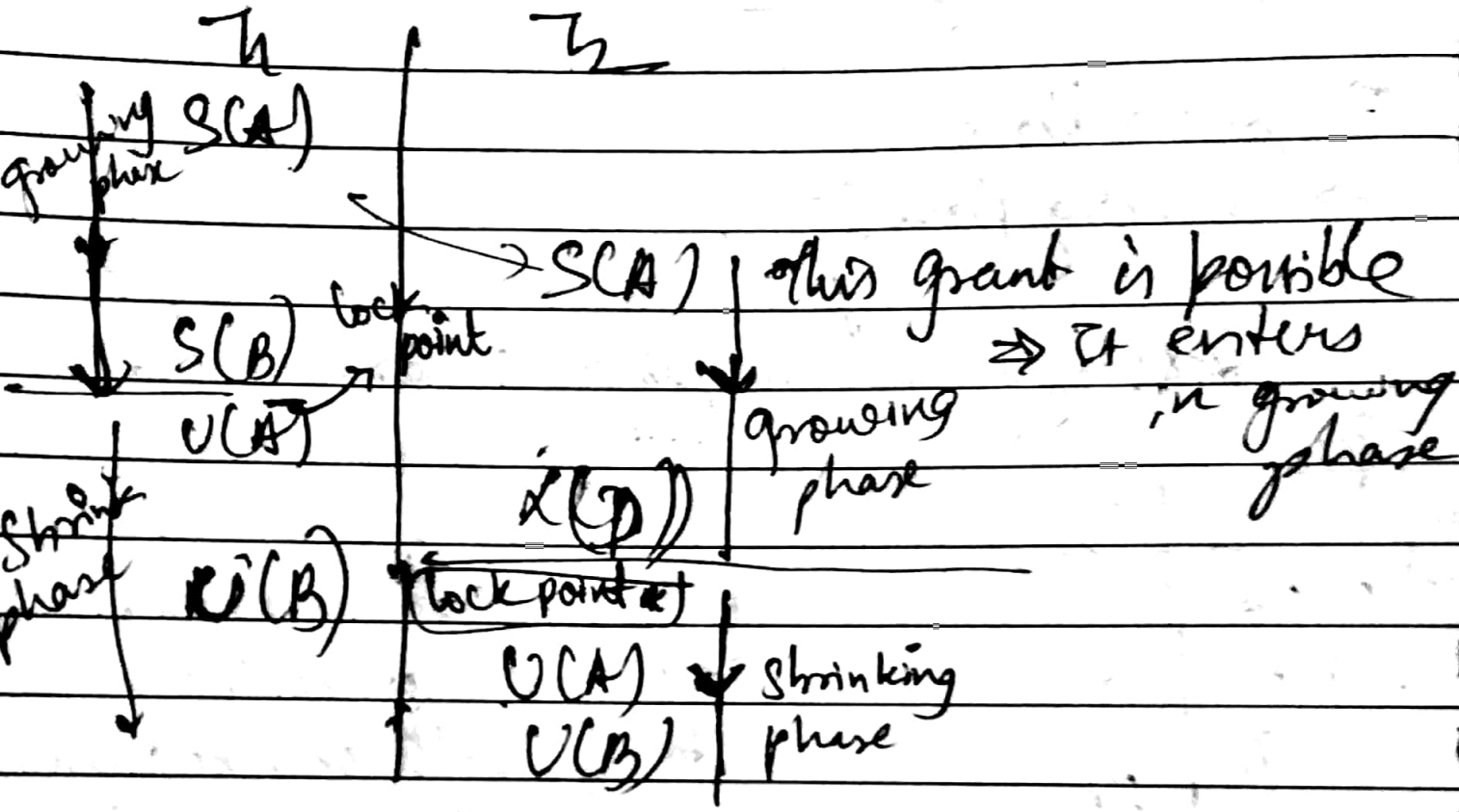
Eg:-



When T_1 is in growing phase, S(A) will not be given to T_2. It'll only be given when T_1 enters in shrinking phase. This will ensure serializability ($T_1 \rightarrow T_2$.)
⇒ Consistency maintained

** Note:: 2PL doesn't mean only one trans^n will run. Even T_2's growing phase can start provided the locks which need to be granted comply with each other i.e.

see example →



Lock point :- where first unlock happens

Schedules the serial seqⁿ of schedule will be the one having lock point earlier
i.e. $T_1 \rightarrow T_2$

Drawbacks of 2PL

Advantages:- Always ensures serializability

Drawbacks:- 1) May not be free from recoverability

2) Not free from ~~deadlocks~~ Cascading rollback

3) Not free from deadlock

T_1	T_2	
$x(A)$		
growing phase	$x(B)$	growing phase
$x(B)$		
	$x(A)$	

4) Not free from starvation

Strict 2PL :- It should satisfy basic 2PL & all exclusive locks should hold until commit/abort.

→ →

Ex

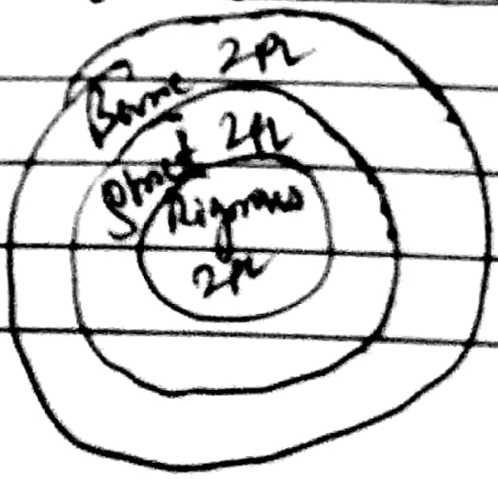
	T_1	T_2
	R(A)	
rollback	R(A)	
	W(A)	
		S(A)
		R(A)

this rollback will not affect T_2.

when commit only then S(A) granted. ⇒ It removes the problem of irrecoverability & cascading rollback as well.

∴ After Strict 2PL, we'll get 1) Strict recoverable
2) Cascadeless

~~Rigorous 2PL~~:- But in Strict 2PL as well, there's deadlock & starvation.



{A→B, B→C}

(AB), (BC)

Rigorous 2PL:- It should satisfy the basic 2PL and all shared, exclusive locks should hold until commit/abort.

~~If~~ These transactions can be serialized in the order in which they commit.

Timestamp Wait-die & wound wait

Assume T_i requests for an item currently held by T_j.

Wait die :- If T_i is older than T_j
→ T_i will wait for T_j
else if T_i is younger
→ T_i will die & restart with same time stamp.