DBMS Unit-4

* Relational algebra :-

→ It is a procedural query lang. It gives step-by-step process to obtain the result of query.

→ Uses operators to perform queries.

* Types of Relational operation :-

1. Select operation :- denoted by sigma (σ)

→ The select operation selects tuples that satisfy a given predicate.

→ Notation :- $\sigma_{p}(r)$

where σ = used for selection prediction

r = used for relation

p = used as propositional logic which may use connector like AND, OR, NOT. These relations can use as relational operators like $=, \neq, \geq, <, >, \leq$

e.g, LOAN Relation

Branch-name	LOAN_NO	Amount
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-12	900
Perryride	L-11	1300

Input :-

$\sigma_{BRANCH\_NAME = "perryride"}(LOAN)$

Output :-

Branch-name	LOAN_NO	Amount
Perryride	L-15	1500
Perryride	L-14	1300

2. Project operation :-

→ This operation shows the list of those attributes that we wish to appear in the result. Rest attributes are eliminated from table.

→ Denoted by ∏

→ Notation :- ∏ A1, A2, An (r)

where A1, A2, .. An = used as an attribute
r = name of relation

→ e.g, CUSTOMER Relation

Name	street	city
Smith	Main	London
Hays	North	Australia
Johnson	Main	Australia
Brooks	North	Polland

Input :-

∏ NAME, CITY (CUSTOMER)

Output :-

Name	City
Smith	London
Hays	Australia
Johnson	Australia
Brooks	Polland

3. Union operation :-

→ The union operation contains all the tuples that are either in R or S or both in R & S

→ Eliminates duplicate tuples

→ R & S must have same no. of attributes.

→ Notation :- R ∪ S

e.g., DEPOSITOR Relation

customer_name	Account_no
Ruja	A-101
Ritu	A-121
Saloni	A-321
Sonam	A-176
Snehalata	A-172

BORROW Relation

customer_name	Account_no
Sneha	L-17
Ruja	L-23
Saloni	L-15
Smith	L-14
Snehalata	L-11

Input :- Π CUSTOMER-NAME (BORROW) $\cup$ Π CUSTOMER-NAME (DEPOSITOR)

Output :-

CUSTOMER_NAME

Ruja
Saloni Ritu
Snehalata Saloni
Sonam
Snehalata
Sneha
Smith

4. Set Intersection :-

→ Contains all the tuples that are in both R & S.

→ Denoted by $\cap$

→ Notation :- R ∩ S
→ Using example from union
→ Input :-
∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output :-
Riya
Kalori
Snehalata

5. Set difference :-
→ Two tuples R and S. The set diff. operation contains all tuples that are in R but not in S.
→ Denoted by (−)
→ Notation R − S
→ Input :-
∏ CUSTOMER_NAME (BORROW) − ∏ CUSTOMER_NAME (DEPOSITOR)
→ Output :-
customer_name
Sneha
Smith

6. Cartesian Product :-
→ Used to combine each row in one table with each row in other table.
→ Also called cross product
→ Denoted by X
→ Notation :- E X D

e.g. EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input :- EMPLOYEE × DEPARTMENT

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename operation :-

→ Used to rename the output relation.

→ Denoted by rho (ρ)

→ e.g. we can use rename operator to rename STUDENT relation to STUDENT 1

ρ (STUDENT 1, STUDENT)

* Join Operation :-

→ A join operation combines related tuple from different relations if and only if a given join condition is satisfied.

→ Denoted by ⋈

→ e.g., EMPLOYEE

EMP_CODE	EMP_NAME
101	Smith
102	Jack
103	Harry

SALARY

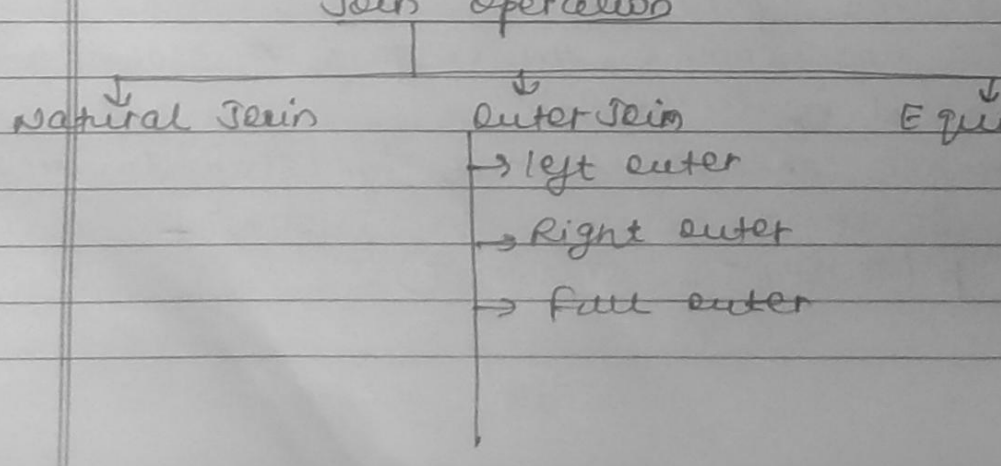
EMP_CODE	SALARY
101	50000
102	30000
103	25000

Operation → (EMPLOYEE ⋈ SALARY)

Result :

EMP_CODE	EMP_NAME	SALARY
101	Smith	50000
102	Jack	30000
103	Harry	25000

→ Types of Join Operation :-



1. Natural Join :-

→ Set of tuples of all combinations in R and S that are equal on their common attribute names

→ Denoted by ⋈

→ Input :-

Π EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output :-

EMP_NAME	SALARY
Smith	50000
Jack	30000
Harry	25000

2. Outer Join :-

→ Used to deal with missing info.

e.g.,

EMPLOYEE

emp_name	STREET	CITY
Ram	Civil-Line	Mumbai
Shyam	Park Street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

emp_name	BRANCH	salary
Ram	Infosys	10000
Shyam	Wipro	20000
Kubet Ravi	HCL	30000
Hari	TCS	50000

Input :- (EMPLOYEE ∝ PART-WORKER)

Output :-

EMP-NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil-Line	Jyoys	Mumbai	10000
Shyam	Park Street	Wipro	Kolkata	20000
Hari	Nehru Nagar	HCL	Hyderabad	50000

* An outer join is basically of 3-types
(a) left outer join
(b) Right " "
(c) Full " "

(a) Left outer join :-
Denoted by :- ⟕

emp-name	Street	City	Branch	Salary
Ram	Civil-Line	Mumbai	Jyoys	10,000
Shyam	Park-Street	Kolkata	Wipro	20,000
Rani	M.G. Street	DELHI	NULL	NULL
Hari	Nehru Nagar	Hyderabad	TCS	50,000

(b) Right outer join :-
Denoted by :- ⟖

emp-name	Street	City	Branch	Salary
Ram	E.L.	Mumbai	Jyoys	10,000
Shyam	P.T.	Kolkata	Wipro	20,000
Kuber	NULL	NULL	HCL	30,000
Hari	Nehru Nagar	Hyd.	TCS	50,000

(c) Full Outer Join :-

→ Tuples in R have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.

→ Denoted by ⟗

→ e.g,

emp-name	Street	City	Branch	Salary
Ram	C.L.	Mumbai	Infosys	10000
Shyam	P.S.	Kolkata	Wipro	20000
Hari	N.S.	Hyderabad	TCS	50000
Rani	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi Join :-

→ also called inner join

→ It is based on matched data as per the equality cond^{n}. The equi join uses comparison operator (=

→ e.g, CUSTOMER Relation

class_id	Name
1	John
2	Harry
3	Jackson

PRODUCT

Product_id	City
1	Delhi
2	Mumbai
3	Noida.

(CUSTOMER ⋈ PRODUCT)

Output :-

Cust_id	Name	Product_id	City
1	John	1	Delhi
2	Harry	2	Mumbai
3	Jackson	3	Noida.

* Tuple Relational Calculus (TRS) :-

→ TRS is a non-procedural query lang. unlike relational algebra.

→ TRS only provides description of the query but it does not provide methods to solve it. ie, explains what to do but not how to do.

→ In TRS, a query is expressed as

$\{ t \mid P(t) \}$

where t = resulting tuples

P(t) = known as predicate and these are cond^n that are used to fet th t

∴ It generates set of all tuples t, such that Predicate P(t) is true for t.

→ P(t) have various cond^n logically combined OR (V), AND (∧), NOT.

also uses quantifiers "there exists", "for all"

→ example :-

Table-1 customer

Customer_name	Street	City
Saurabh	A 7	Patiala
Mehak	B 6	Jalandhar
Sumiti	D 9	Ludhiana
Ria	A 5	Patiala

Table-2 : Branch

Branch-name	Branch-city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-3 : Account

Account-no.	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table-4 : Loan

Loan-no.	Branch-name	Amt.
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table-5 : Borrower

Customer-name	Loan-no.
Saurabh	L33
Mehak	L49
Ria	L98

Table-6 : Depositor

Customer-name	Account-no.
Saurabh	1111
Mehak	1113
Sumiti	1114

ANKIT
Date
Page

Queries-1 :- Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$\{t \mid t \in loan \wedge [amount] >= 10000\}$

Loan_number	Branch_name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

Queries-2 :- Find the loan number for each loan of an amount greater or equal to 10000

$\{t \mid \exists\ s \in loan\ (t[loan\ number] = s[loan\ number] \wedge s[amount] >= 10000)\}$

Loan number
L33
L35
L98

* Pitfall in Relational Database design

→ A bad design of several properties including
- Repetition of info.
- Inability to represent certain info.
- loss of info.

→ LENDING SCHEMA

branch-name	branch-city	assets	customer name	loan no	amount
Downtown	London	1,90,000	Jones	L-17	1000
Perryride	Australia	2,00,000	Smith	L-23	3000
Redwood	Australia	1,50,000	Jackson	L-29	5000
Downtown	Portland	1,90,000	Jackson	L-11	2000
Perryride	Australia	2,00,000	Glenn	L-16	22000

→ Repeating info. wastes space and complicates updating the databases.
For eg, the assets of perryride branch changes from 2,00,000 to 3,00,000. Under the design many tuples of the lending relation needs to be changed.
We ensure that every tuple consists to the perryride branch is updated, else our database will show two different assets values for perryride branch.

→ Another problem with lending schema is that we cannot represent directly the info. concerning a branch (branch-name, branch-city, assets) unless there exists atleast one loan at the branch. Because tuples in lending relation requires values for loan-no., amount and customer value.

→ Solution to these problems :-
(i) Introduction of NULL values but null values are difficult to handle
(ii) we create branch info only when first loan appln at that branch is made. (worse, we have to delete this info. when loan is paid)
↳ clearly this situation is undesirable
→ branch info. would be available regardless of whether or not loans are currently maintained

* Decomposition :-

→ Process of breaking down relation into multiple relation.

→ It should be lossless and if there is no proper decomposition of relation then it may lead to problems like loss of info.

→ Decomposition helps in removing some of the problems of bad design such as redundancy, inconsistencies.

→ 2 - types of decomposition :-

(i) Lossy :-

→ The decomposition of R into R1 and R2 is lossy when the join of R1 and R2 doesn't yield the same relation as in R.

e.g, Student

Roll_no.	S-name	department
111	Parimal	Computer
222	Parimal	Electrical

This relation is decomposed into two relations no_name and name_department

Roll_no.	S-name
111	Parimal
222	Parimal

S-name	dept
Parimal	Computer
Parimal	Electrical

In lossy decomposition, spurious tuples are generated when a natural join is applied to the relations in decomposition

Spurious tuples → extra tuples (rows) which occur as a result of joining two tables in wrong manner.

Roll-no	S-name	dept	
111	Parimal	Computer	bad or loss decomp^n
111	Parimal	Electrical	
222	"	Computer	
222	"	Electrical	

(ii) Lossless join decomposition :-

→ The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 yield the same relation in R.

→ always defined w.r.t specific set of dependencies

e.g.,

Stud-name

Roll-no	S-name
111	Parimal
222	Parimal

Stud-dept

Roll-no	dept
111	Computer
222	Electrical

Stud-joined

Roll-no	S-name	dept
111	Parimal	Computer
222	Parimal	Electrical

No spurious Tuple is generated

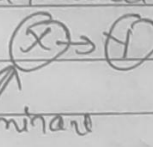
* Properties of decomposition :-

(i) Wires guarantee that the join will result in the same relation as it was decomposed.

(ii) Lack of data redundancy :- The proper decomposition should not suffer from any data redundancy. Lack of data redundancy is also known as repetition of info

(iii) Dependency Preservation :-
Every dependency must be satisfied by atleast one decomposed table.
e.g., Relation R (A, B, C, D) with functional dependency set (A → BC).
Relation R is decomposed into R1 (A BC) and R2 (A D) which is dependency preserving because F.D. A → BC is a part of relation R1 (A BC)

* Functional Dependency :-
is a method that describes the relationship bet^n the attributes dependent



e.g., Sid → Sname

1	Rajit
1	Rajit

valid

1	→ Rajit
2	→ Ramjit

valid

1	Rajit
2	Varun

valid

Sid	Sname
1	Rajit
1	Varun

Invalid case

Sid says one student but Ranjit & Varun are 2-different students

* FD is of 2-type :-
(1) Trivial (2) Non-Trivial

(1) Trivial F.D :-
X → Y then Y is subset of X
→ always true / valid
→ coz what we are determining is already subset of left hand side.

We always check valid is valid in case of non-trivial F.D. because Trivial is a kind of Reflexive which is always valid

$X \to Y$

L.H.S $\cap$ R.H.S $\neq \phi$

e.g., Sid Sname $\to$ Sid

Sid Sname $\cap$ Sid = Sid i.e, $\neq \phi$

(ii) Non-Trivial :-

$X \to Y$ then Y is not subset of X

L.H.S. $\cap$ R.H.S $= \phi$

e.g, Sid $\to$ Sname
Sid $\to$ Sphone no.

} we need to check the case

* Properties of F.D. :-

(i) Reflexivity :- if Y is subset of X then $X \to Y$ (always valid)

(ii) Augmentation :- if $X \to Y$ then $XZ \to YZ$

e.g, Sid $\to$ Sname

Sid Sphone $\to$ Sname Sphone

this will also be valid

(iii) Transitive :- if $X \to Y$ and $Y \to Z$ then $X \to Z$

Sid $\to$ Sname Sname $\to$ Scity

$\Rightarrow$ Sid $\to$ Scity

(iv) Union :- if $X \to Y$ and $X \to Z$ then $X \to YZ$

(union)

(v) Decomposition :- if $X \to YZ$ then $X \to Y$ and $X \to Z$

Note :- $XY \to Z$ $X \to Z$, $Y \to Z$

Invalid coz ~~we are to~~ X and Y are effectively used to determine Z and Hence, cannot be broken

(vi) Pseudotransitivity :-
if $X \to Y$ and $WY \to Z$ then $WX \to Z$

(vii) Composition :-
if $X \to Y$ and $Z \to W$ then $XZ \to YW$

* Closure Method :-
helps to find all the candidate keys in the table that are possible

e.g. R (A B C D)
FD { A → B, B → C, C → D }
$A^+ = BCDA$ Prime attribute = { A }
$B^+ = BCD$ Non " = { B C D }
$C^+ = CD$
$D^+ = D$
Candidate key = { A }
Note :- $(AB)^+ = ABCD$
AB = candidate key ✗ (cannot be candidate key as candidate key needs to be minimal)
↑ anything associated with candidate key will make it superkey
∴ AB = ~~a~~ superkey.

R (A B C D)
FD = { A → B, B → C, C → D, D → A }
$A^+ = ABCD$
$B^+ = BCDA$
$C^+ = CDAB$
$D^+ = DABC$
C.K. = { A, B, C, D }

Prime attribute ⇒ used in making of C.K.

Prime attribute = { A, B, C, D }

Non " = ∅ { Φ }

R (A B C D E)

FD = { A → B, BC → D, E → C, D → A }

E = B D C A (E)

↑ which is E ~~can be~~ would be on right side and ~~also~~ on left side

Each and every candidate key must contain E

E^+ = E C { E is not the only candidate key but is used to finding the candidate keys }

AE^+ = A B E C D

BE^+ = B E C D A

CE^+ = C E A

DE^+ = D E A B C

C.K = { AE ; DE , BE }

↓

DE^+ = D E A B C

* Normalization :-

→ It is a technique to remove or reduce redundancy from a table.

Row level duplicacy can be removed by using primary key (unique + not null)

Sid	Sname	Sage
1	Ram	20
2	Varun	25
1	Ram	20

ANKIT

Date
Page

course-id → course-name → faculty-id

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C_1	DBMS	P_1	John	80 000
2	Ravi	C_2	Java	F_2	Bob	40000
3	Nith	C_1	DBMS	F_1	John	30000
4	Amit	C_1	DBMS	F_1	John	30000

5	varun	C_{10}	MBBS

we cannot insert this directly as it is only said that a new course is introduced without telling about the student

3 types of Problem

① Insertion anamaly

→ cannot add dumny value or special ch.

⇒ cannot insert directly as it will create problem.

② Deletion anamaly.

⇒ we want to ~~delete~~ remove the db of roll 2

Delete from student
where SID = 2

∴ it will delete whole row by only removing the SID which means extra info. is deleted which cannot be recovered

③ updation anamaly

eg, we want the name of Amit as Amit Pal

update studen
Set Sname = 'Amit Pal'
where SID = 1

Now, if we want to change salary of F_1 from 30K to 40K

∴ Due to column level duplicacy the salary of all the F_1 changes from 30K to 40K

Solution of Normalization :-

Dividing the table.

Sid	Sname

Cid	Cname

Fid	Fname	Salary

* First Normal Form :-

→ Table should not contain ~~single~~ any multi-valued attribute.

eg)

Roll-no.	Name	course	Not in 1NF
1	Sai	C/C++	
2	Harsh	Java	
3	Omkar	C/DBMS	

(•) It can be converted into 1NF by 3- ways

(1)

Roll-no	Name	course
1	Sai	C
1	Sai	C++
2	Harsh	Java
3	Omkar	C
3	Omkar	DBMS

Primary key Roll-no course combining two also called composite Primary key

(2)

Roll-no.	Name	course 1	course 2	. . .
1	Sai	C	C++	. . . .
2	Harsh	Java	Null	Null . . . .
3	Omkar	C	DBMS	

Primary key = Roll-no.

Note :- there can be ~~no~~ no. of courses in which a student can be enrolled and on the other hand one student can be enrolled only in 1 course so there will be more no. of nulls which is not a good representation

③ By dividing the table foreign key

Roll-no	Name
1	Sai
2	Harsh
3	Omkar

Base Table
as here Roll-no is primary key.

Roll-no	Course
1	C
1	C++
2	Java
3	C
3	DBMS

Primary key (Roll-no, Course)
Foreign key (Roll-no) which takes reference from Roll-no of Base table

* 2 NF :- is essential key as there is no incoming edge on them and nobody can find them they must already be a part of candidate key

R (A B C D)
FD (A → D , B → D , B → C)
$(AB)^+ = A\ B\ C\ D$
↑ candidate key.

Prime attribute {A, B}
Non " {C, D}

Second Normal Form

For a table to be in the Second Normal Form, it must satisfy two conditions—

1) The table should be in the 1NF.
2) There should be no Partial Dependency.

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

$AB \rightarrow D$

$B \rightarrow C$

$(AB)^+$ is a candidate key

$A, B \in$ prime attributes

$C, D \in$ non-prime attributes

$B \rightarrow C$ is a partial dependency when the non-prime attribute is depend on the part of a candidate key.

So it is not in 2NF.

we have to convert in 2NF.

$R(ABCD)$

$R_1(ABD)$

AB is C.K.

$R_2(\overrightarrow{B}\overrightarrow{C})$

B is a C.K.

How to identify it?
It is in 2 NF.

Q → R (A B C D E)

AB → C (P.D.)

D → E (P.D.)

R (A B C D E)

(ABD)$^+$ = ABDCE

So (ABD)$^+$ is a candidate key.

So it is not in 2 NF.

R (A B C D E)

- R_1 (AB C)
- R_2 (D.E)
- R_3 (A B D)

More detailed discussion on 2NF

R(A B C)

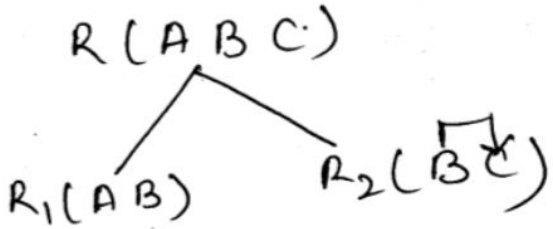
$B \rightarrow C$

so $(AB)^+$ is a candidate key

A, B ∈ prime attribute

C ∈ Non Prime attribute

so it has partial dependency.



A	B	C
a	1	x
b	2	y
a	3	2
c	3	z
d	3	z
e	3	z

R1

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

R2

B	C
1	x
2	y
3	z
3	z
3	z
3	z

* 3NF :-

→ Table or relation must be in 2NF

→ there should be no transitive dependency in table

Roll_no	state	city
1	Punjab	Mohali
2	Haryana	Ambala
3	Punjab	Mohali
4	Haryana	Ambala
5	Bihar	Patna

C.K = { Roll_no }

F.D. = Roll_no. → state state → city

Roll_no. → state : valid as Roll_no is candidate key or prime attribute is used to determine non-prime attribute

state → city : here is the problem where non-prime attribute is determining non-prime attribute which should be not to there

Non-Prime attributes = { state, city }

Now, Roll no. → city not given in F.D. but because of transitive property Roll no. → city through state which is non-prime attribute

e.g., R(A B C D)

F.D. = AB → C, C → D

C.K : AB AB+ : A B C D

P.A = A, B

N.P.A. = C, D

C.K. = candidate key
S.K. = super key

e.g., R(A B C D)

F.D : AB → CD , D → A

C.K : {AB, DB}

AB^+ = ABCD

↑ check whether A is on Right side

replace A by D

DB^+ = DBAC

P.A. = {A, B, D}

N.P.A. = {C}

To check Transitivity, for each FD

LHS must be C.K. or S.K (or)

RHS must be prime attribute

F.D. AB → CD , D → A

↑ as AB = C.K ↑ Prime attribute

∴ This table is in 3NF

* B-C NF (Boyce Codd Normal form)

→ also called special case of 3NF

→ L.H.S. of each F.D. should be C.K. or S.K

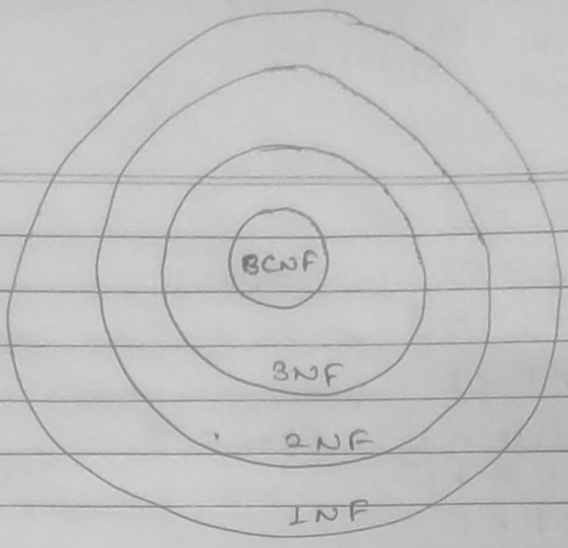
→ e.g.,

Roll-no.	Name	voter_id	age
1	Ravi	K0123	20
2	Varun	M034	01
3	Ravi	K786	23
4	Rahul	D286	21

C.K. = {Roll no., voter id}

F.D. = { Roll no. → name ✓
Roll no. → voterid ✓
voterid → age ✓
voterid → Roll no. ✓ } valid as L.H.S = C.K

∴ Table is in B-C NF



* 3NF always ensures 'Dependency Preserving Decomposition but not in BCNF
Both 3NF & BCNF ensures lossless decomposition

e.g, R(A B C D)

$AB \rightarrow CD$, $D \rightarrow A$

$AB^+ = ABCD$ $\quad$ $CB^+ = CB$ ×

$DB^+ = DBAC$

C.K. = {AB, DB}

P.A. = {A, B, D}

N.P.A. = {C}

$AB \rightarrow CD$ $\quad$ $D \rightarrow A$

3NF

Not BCNF as D is not C.K.

R(A B C D)

$D^+ = DA$

$R_1(\underline{D}A)$ $R_2(B\ C\ \underline{D})$

Lossy Join Decomposition

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

$R_1(AB)$ $R_2(BC)$

R_2

B
2
2
3

B	C
2	1
2	2
3	2

we have common B as we can merge or join these two tables in future if we want.

value of C if the value of A = 1'
C is in R_2 and A is in R_1 so we not directly find the value we need to merge the both the tables.

2. C from R_2 Natural Join R_1 where R_1. A = 1'

product of R_1 & R_2 + some cond^

B	C
2	1 ✓
2	2 ✓
3	2
2	1 ✓
2	2 ✓
3	2
2	1
2	2

check the equivalence of two common attributes i.e, where they are having equal values.

after joining → common value are merged into one

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

not in original called spurious tuple

called lossy decomposition as the original table had 3-tuples and we divided it in 2-tables and after joining 2-tables the new table has 5-tuples

called lossy ⇒ there is duplicacy / inconsistency which we are calling lossy

(1) Common attribute should be C.K. or S.K of either R_1 or R_1 or Both
∴ In above example B & C cannot be common attribute
only 'A' can be common attribute which has unique values.
i.e. $R_1(AB)$ $R_2(AC)$

(2) (a) $R_1 \cup R_2 = R$
$AB \cup AC = ABC$
(b) $ABC = ABC$

(3) (a) $R_1 \cap R_2 \neq \phi$
$AB \cap AC \neq \phi$
$A \neq \phi$

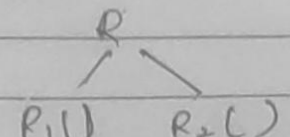
(3) (b) R_1

* Dependency Preserving Decomposition :

R (A B C D)

FD A → B, B → C

FD+ A → C (From transitive property)



Functional dependency also get divided

FD1 FD2

FD1 ∪ FD2 = FD*

e.g. let R(A B C D) with F.D.

{ A → B, B → C, C → D, D → B }

R is decomposed into R1(A B), R2 (B C), R3(B D)

R1 (A B)	R2 (B C)	R3 (B D)
Trivial { A → A ✗	B → C ✓	B+ = BCD B → D ✓
Trivial will not take { B → B	C → B ✓	D → B ✓
A → B ✓	C+ = C D B	
B → A ✗		
B+ = B C D		

✓ A → B
✓ B → C
C → B
B → D
D → B

Now, we will check that whether we can derive { A → B, B → C, C → D, D → B } from this F.D.

A → B, B → ✗, C → D, D → B

C+ = C B D

∴ all 4 F.D are preserved.

* 2NF :-

→ In 1NF

→ No partial dependency

→ Only full dependency

(AB) → C ✓valid
non-unique

B → C
A → C
} partial dependency which is not allowed in 2NF

* 4NF :-

→ In BCNF

→ No Multivalued dependency
X ↠ Y

e.g. varun → 3 ~~Phone~~ Mobile no.
→ 3 Mail

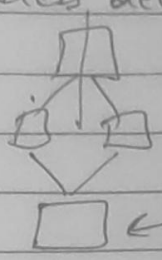
varun	M1	E1
varun	M2	E2
"	M1	E3
	M2	E1
	M2	E2
	!	
	!	

3 rows will be created as varun is multivalued dependent on ~~Phone~~ Mobile no. and E-mail And ∵ Mobile no. & E-mail has no relation so, we should make ~~dept~~ seperate table

* 5NF :-

→ 4th 4NF

→ Lossless decomposition



← to be the original table the common attribute must be C.K.