

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 :- $\sigma_P(\tau)$

where σ = used for selection predicate

τ = 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
Pereyende	L-15	1500
Downtown	L-14	1500
Mirrus	L-13	500
Roundhill	L-12	900
Pereyende	L-11	1300

Input :-

$\sigma_{\text{BRANCH-NAME} = "Pereyende"} (\text{LOAN})$

Output :-

Branch-name	LOAN-NO	Amount
Pereyende	L-15	1500
Pereyende	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 Π
- Notation :- $\Pi A_1, A_2, \dots A_n (\tau)$
where $A_1, A_2, \dots A_n$ = used as an attribute
 τ = name of relation
- e.g., CUSTOMER Relation

Name	Street	City
Smith	Main	London
Hays	North	Australia
Johnson	Main	Australia
Brooks	North	Polland

Input :-

$\Pi \text{NAME, CITY } (\text{CUSTOMER})$

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 :- $R \cup S$

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
Sonam	L-14
Snehalata	L-11

Input :- \cap CUSTOMER-NAME (BORROW) \cup
 \cap CUSTOMER-NAME (DEPOSITOR)

Output :-

CUSTOMER-NAME
Ruja
Saloni Ritu
Snehalata Saloni
Sonam
Snehalata
Sneha
Sonam

Q: Set Intersection :-

- Contains all the tuples that are in both R & S.
- denoted by \cap

→ Notation :- R ∩ S

→ Using example from union

→ Input :-

$$\sqcap \text{ CUSTOMER-NAME (BORROW)} \sqcap \text{ CUSTOMER-NAME (DEPOSITOR)}$$

Output :-

Riya

Laloni

Snehalata

5. Set difference :-

→ Two tuples R and S. The set diff. operation contains all ~~the~~ tuples that are in R but not in S.

→ denoted by (-)

→ notation R - S

→ Input :-

$$\sqcap \text{ CUSTOMER-NAME (BORROW)} - \sqcap \text{ CUSTOMER-NAME (DEPOSITOR)}$$

→ 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

<u>EMP-ID</u>	<u>EMP-NAME</u>	<u>EMP-DEPT</u>
1	smith	A
2	Harry	C
3	John	B

DEPARTMENT

<u>DEPT-NO</u>	<u>DEPT-NAME</u>
A	marketing
B	Sales
C	Legal

Input :- EMPLOYEE X DEPARTMENT

<u>EMP-ID</u>	<u>EMP-NAME</u>	<u>EMP-DEPT</u>	<u>DEPT-NO</u>	<u>DEPT-NAME</u>
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 `RENAME`
- e.g. we can use `RENAME` operator to rename STUDENT relation to STUDENT_L
 $\text{RENAME STUDENT AS STUDENT_L}$
 $\text{P (STUDENT_L, STUDENT)}$

* Join Operations :-

- A join operation combines related tuples from different relations if and only if a given join condition is satisfied.
- denoted by \bowtie
- e.g., EMPLOYEE

EMP-CODE	EMP-NAME
101	Smith
102	Jack
103	Harry

SALARY

EMP-CODE	SALARY
101	50000
102	30000
103	25000

operation \rightarrow (EMPLOYEE \bowtie SALARY)

Result :-

EMP-CODE	EMP-NAME	SALARY
101	Smith	50000
102	Jack	30000
103	Harry	25000

→ Types of Join Operations :-

Join Operations

↓
natural join

↓
Outer Join

↓
Equi Join

→ left outer

→ Right outer

→ full outer

1. Natural Join :-

- set of tuples of all combinations in R and S that are equal on their common attribute names
- denoted by \bowtie
- Input :-

$\Pi_{EMP_NAME, SALARY} (EMPLOYEE \bowtie SALARY)$

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	Kerala
Rani	H.C. Street	Delhi
Hari	Nehru nager	Hyderabad

FACT-WORKERS

emp-name	BRANCH	salary
Ram	TCS	10000
Shyam	Wipro	20000
Rani	HCL	30000
Hari	TCS	50000

Input :- (EMPLOYEE & FALT-WORKERS)

Output :-

<u>EMP-NAME</u>	<u>STREET</u>	<u>CITY</u>	<u>BRANCH</u>	<u>SALARY</u>
Ram	Civil-LiB	Gyads	Mumbai	10000
Ishyam	Park-Street	Wifero	Kollata	20000
Hari	Nehru Nagar	HCL	Hyderabad	50000

- * An outer join is basically of 3-types
 - left outer join
 - right " "
 - full " "

(a) Left outer join :-

denoted by :- 

<u>emp-name</u>	<u>street</u>	<u>city</u>	<u>Branch</u>	<u>Salary</u>
Ram	Civil-LiB	Mumbai	Gyads	10,000
Ishyam	Park-Street	Kollata	Wifero	20,000
Ram	M.G. Street	Delhi	NULL	NULL
Hari	Nehru Nagar	Hyderabad	TCS	50,000

(b) Right outer join :-

denoted by :- 

<u>emp-name</u>	<u>street</u>	<u>city</u>	<u>Branch</u>	<u>Salary</u>
Ram	E.L.	Mumbai	Gyads	10,000
Ishyam	P.T.	Kollata	Wifero	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.,

<u>emp-name</u>	<u>street</u>	<u>city</u>	<u>Branch</u>	<u>salary</u>
Ram	C.L.	Mumbai	Infosys	10000
Shyam	P.S.	Kolkata	Wipro	20000
Hari	N.S.	Hyderabad	TCS	50000
Rani	H.H. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	MLL	30000

3. Equi Join :-

→ also called inner join

→ It is based on matched data as per the equality condⁿ. The equi join uses comparison operator (=)

→ e.g., CUSTOMER Relation

<u>class-id</u>	<u>Name</u>
1	John
2	Harry
3	Jackson

PRODUCT

<u>Product-id</u>	<u>City</u>
1	Delhi
2	Mumbai
3	Nei'da

(CUSTOMER & PRODUCT)

Output :-

Class-id	Name	Product-id	City
1	John	1	Delhi
2	Harry	2	Mumbai
3	Jackson	3	Noida

* Tuple Relational Calculus (TRC) :-

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

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

→ In TRC, a query is expressed as

{ t | P(t) }

where t = resulting tuples

P(t) = known as predicate and these are condⁿ that are used to fetch t

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

→ P(t) have various condⁿ logically combined OR (V), AND (Λ), NOT

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

→ example :-

Table-1 customer

Customer-name	Street	City
Laurabh	A 7	Patiala
Mehak	B 6	Jalandhar
Sunita	D 9	Ludhiana
Ria	A 5	Patiala

Table-2 : Branch

<u>Branch-name</u>	<u>Branch-city</u>
A BC	Patiala
DEF	Ludhiana
MHI	Jalandhar

Table-3 : Account

<u>Account-no.</u>	<u>Branch name</u>	<u>Balance</u>
1111	A BC	50000
1112	DEF	10000
1113	MHI	9000
1114	A BC	7000

Table-4 : Loan

<u>Loan-no.</u>	<u>Branch-name</u>	<u>Amnt.</u>
L33	A BC	10000
L35	DEF	15000
L49	MHI	9000
L98	DEF	65000

Table-5 : Borrower

<u>Customer-name</u>	<u>Loan-no.</u>
Saurabh	L33
Mehak	L49
Ria	L98

Table-6 : Depositor

<u>Customer-name</u>	<u>Account-no.</u>
Saurabh	1111
Mehak	1113
Sumiti	1114

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

$$\{t \mid t \in \text{loan} \wedge [\text{amount}] \geq 10000\}$$

<u>loan-number</u>	<u>Branch-name</u>	<u>Amount</u>
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

$$\{t \mid \exists s \in \text{loan} (t[\text{loan number}] =$$

$$s[\text{loan number}] \wedge \\ s[\text{amount}] \geq 10000\})$$

loan number

L33

L35

L98

* Pitfall in Relational Database Design

- A bad design of several properties including
 - Repetition of 'yo'.
 - Inability to represent certain 'yo'.
 - loss of 'yo'.

→ LENDING SCHEMA

<u>branch-name</u>	<u>branch-city</u>	<u>assets</u>	<u>customer-name</u>	<u>loan-ID</u>	<u>amount</u>
Downtown	London	1,80,000	Jones	L-19	1000
Perryridge	Australia	2,00,000	Smith	L-23	3000
Redwood	Australia	1,80,000	Taylor	L-29	5000
Mountain	Petland	1,90,000	Galloway	L-11	2000
Perryridge	Australia	2,00,000	Wren	L-16	22000

→ Repeating info wastes space and complicates updating the database.

For e.g., the assets of perryeide 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 perryeide branch is updated, else our database will show two different assets values for perryeide 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 at least one loan at the branch. Because tuples in lending relation requires values for loan-no., amount and customer value.

→ Solution to these problems :-

(i) Instead of NULL values but null values are difficult to handle

(ii) we create branch info only when first loan appliⁿ 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 relations.
- It should be lossless and if there is no perfect decomposition of relation then it may lead to problem like loss of info.
- Decomposition helps in removing some of the problems of bad designs such as redundancy, inconsistency.
- 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	S-name	dept
111	Parimal	Parimal	computer
222	Parimal	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 ^d
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 as R.
- always defined w.r.t. specific set of dependencies
e.g.,

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

stud-joined

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

no spurious tuple is generated

* Properties of decomposition :-

- (i) gives 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 at least one decomposed table.

e.g., Relation R (A, B, C, D) with functional dependency set ($A \rightarrow BC$).

Relation R is decomposed into R1(A BC) and R2(AD) which is dependency preserving because F.D. $A \rightarrow BC$ is a part of relation R1(A BC)

* Functional Dependency :-

is a method that describes the relationship between the attributes dependent

$X \rightarrow Y$ means X determines Y or Y is determined by X

e.g., Sid \rightarrow Sname

1 Ranjit	$1 \rightarrow$ Ranjit
1 Ranjit	$2 \rightarrow$ Ranjit

valid

valid

Sid Sname

1 Ranjit
2 Varun

Valid

1 Ranjit
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 :-

$X \rightarrow Y$ then Y is subset of X

\rightarrow always True / valid

\rightarrow coz what we are determining is already subset of left hand side.

→ we always check valid or invalid in case of non-trivial F.D. because Trivial is a kind of Reflexive which is always valid ANKIT

$X \rightarrow Y$

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

e.g., Sid Sname \rightarrow Sid

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

(ii) Non-Trivial :-

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

e.g., L.H.S. \cap R.H.S. = \emptyset

e.g., Sid \rightarrow Sname] we need to
Sid \rightarrow Sphone no.] check the case

* Properties of F.D. :-

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

(P) Augmentation :- if $X \rightarrow Y$ then $XZ \rightarrow YZ$

e.g., Sid \rightarrow Sname

Sid Sphone \rightarrow Sname Sphone
this will also be valid

(P) Transitive :- if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Sid \rightarrow Sname Sname \rightarrow Scity

\Rightarrow Sid \rightarrow Scity

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

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

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

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

(vi) Pseudotransitivity :-

$w \rightarrow x \rightarrow y$ and $y \rightarrow z$ then $wx \rightarrow z$

(vii) Composition :-

$x \rightarrow y$ and $z \rightarrow w$ then $xz \rightarrow yw$

* Closure Method :-

helps to find all the candidate keys in the table that are possible

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

$FD(A \rightarrow B, B \rightarrow C, C \rightarrow D)$

$$A^+ = BCD$$

positive attribute - {A}

$$B^+ = BCD$$

non " = {BCD}

$$C^+ = CD$$

$$D^+ = D$$

Candidate key = {A}

Note :- $(AB)^+ = ABCD$

AB = candidate key \times {
 anything associated
 with candidate key
 will make it superkey
 $\therefore AB$ = superkey.

$R(A B C D)$

$FD = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$

$$A^+ = ABCD$$

$$B^+ = BCDA$$

$$C^+ = CDAB$$

$$D^+ = DABC$$

$$C.K. = \{ A, B, C, D \}$$

Primary attribute \Rightarrow used in making of C.K.
 Primary attribute = {A, B, C, D}

Non " = $\emptyset \subseteq \{\Phi\}$

R (A B C D E)

FD = {A \rightarrow B, BC \rightarrow D, E \rightarrow C, D \rightarrow A}

E = B D C A (E)

\uparrow which is
E can be on right side
and also \downarrow should be on left side

Each and every candidate key must contain E

$E^+ = EC$ if 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]

\downarrow
 $DE^+ = DE A B C$

* 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	Age
1	Ram	20
2	Varun	25
1	Ram	20

Date _____
Page _____

ANKIT

SId	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C1	DBMS	F1	John	80000
2	Ravi	C2	Java	F2	Bob	40000
3	Nithi	C1	DBMS	F1	John	30000
4	Amit	C1	DBMS	F1	John	30000

[S] varon

[C10] MBBS

we cannot insert

3-types of Problem

① Insertion anomaly

⇒ cannot add dummy value
or special ch.

⇒ cannot insert directly as it will create
problem.

this directly as it is
only said that a new
course is introduced
without telling
about the student

② Deletion anomaly.

⇒ we want to delete remove the ab of all 2
Delete from Student

where SId = 2

⇒ it will delete whole row by only
removing th SId which means extra
info. is deleted which cannot be recovered

③ updation anomaly

e.g., we want the name of Amit as Amit Pal
update student

Set Sname = 'Amit Pal'

where SId = 1

now, if we want to change salary of F1 from
30K to 40K

⇒ Due to column level duplicacy the
salary of all the F1 change from
30K to 40K

Solution of Normalization :-

So dividing the table:-

Sid	Sname	Cid	Cname	Fid	Fname	Salary
-----	-------	-----	-------	-----	-------	--------

* First Normal Form :-

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

e.g.)	Roll-no.	Name	Course	Not in INF
	1	'Sai'	C/C++	
	2	Harsh	Java	
	3	Omkar	C(DBMS)	

① It can be converted into 1NF by 3 ways

①	Roll-no	Name	course	Pk
	1	sai	c	Roll-no
	1	sai	C++	course
	2	Harsh	Java	combining two
	3	Omkar	c	also called
	3	Omkar	DBMS	(composite Primary key)

②	Roll-no.	Name	course 1	course 2	---
	1	sai	c	C++	---
	2	Harsh	Java	null	null
	3	Omkar	c	DBMS	---

Pk = Roll-no.

Note :- there can be ~~no~~ ^{more} 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	Roll-no.	Course
1	Sai	1	C
2	Marsh	1	C++
3	Omkar	2	Java

Base Table

as here Roll-no is

primary key.

Primary key
(Roll-no, course)

Foreign key (Roll-no)

which takes
reference from
roll-no of Base
table

* 2 NF :- is candidate as there is no
R (A B C D) primary colo on them and nobody
can find them they must
already be a part
PD (A → D, B → D, B → C) of candidate key
 $(AB)^+ = ABCD$

candidate primary attribute {A, B}
key. 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 ∈ prime attributes

C, D ∈ 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.

$R(ABCD)$

$R_1(ABD)$

\Downarrow
AB is C.I.C.

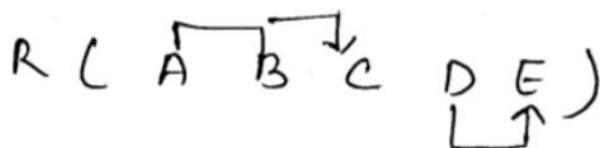
$R_2(B\overline{C})$

\overline{B} is a C.K.

How to identify it?

It is in 2NF.

~~R~~ $\rightarrow R(A B C D E)$
 $A B \rightarrow C$ (P.D.)
 $D \rightarrow E$ (P.D.)



$$(ABD)^+ = ABDCE$$

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

So it is not in 2NF.

$R(A B C D E)$

$R_1(A \overbrace{B}^{\downarrow} C)$

$R_2(D \overbrace{E}^{\uparrow})$

$R_3(A B D)$

More detailed discussion on 2NF

R(A B C)

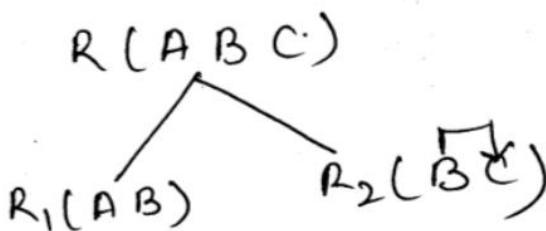
B → C

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	z
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

Reu-no	state	city
1	Punjab	Mohali
2	Haryana	Surbala
3	Punjab	Mohali
4	Haryana	Surbala
5	Bihar	Patna

C.K = { Reu-no }

F.D. = Reu-no. → state state → city

valid as

Reu candidate key
of prime attribute

is used to determine
non-prime attribute

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

Non-prime = { state, city }
attributes

Now, if Reu no. → city

not given in F.D. but because of
transitive property Reu 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



ANKIT

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

F.D.: AB \rightarrow CD, D \rightarrow A

C.K.: {AB, DB}

$AB^+ = AB\ CD$

↑
check whether A is on right side
 D

replace A by D

$DB^+ = DBAC$

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

N.P.A. = {C}

To check Permanently, for each FD

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

RHS must be prime attribute

F.D.: AB \rightarrow CD, D \rightarrow 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

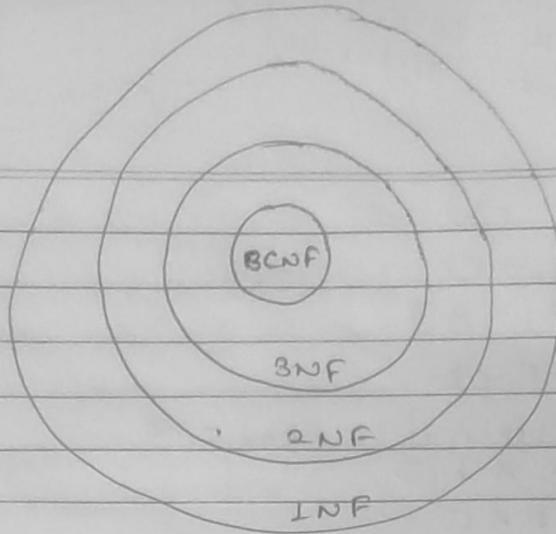
→ L.H.S. of each F.D. should be C.K. or S.K.

→ e.g.,

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

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

F.D. = { Roll no. \rightarrow name ✓
 Roll no. \rightarrow veterid ✓ } valid as
 veterid \rightarrow age ✓ LHS = C.K.
 veterid \rightarrow Roll no. ✓ } \therefore 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)

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

$$AB^+ = ABCD \quad CB^+ = CB \times$$

$$DB^+ = DBAC$$

$$C.K. = \{AB, DB\}$$

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

$$N.P.A. = \{C\}$$

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

3NF

not BCNF as D is not C.K.

R(A,B,C,D)

$$D^+ = DA$$

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

Lucky Join Decomposition

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

(A,B)

R₂(B,C)R₁

B	R ₁	B	C
2		2	1
2		2	2
3		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'
 is in R₂ and A is in R₁ so we
 cannot directly find the value we need
 to merge the both the tables.

2. C from R₂ Natural Join R₁,
 where R₁. A = '1'

product of R₁ & R ~~+ some condition?~~

B	C	check the equivalence of two common attributes i.e., where they are having equal values.
2	1 ✓	
2	2 ✓	
3	2	
2	1 ✓	
2	2 ✓	
3	2	
2	1	
2	2	

2 1 ✓

2 2 ✓

3 2

2 1

2 2

after joining

common value are
merged into one

ANKIT

Date _____

Page _____

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

not in original
called supersede tuple

common value are merged into one

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 \Rightarrow 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_2 or Both
i.e. for above example B & C cannot be common attribute

only 'A' can be common attribute which has unique values.

i.e. $R_1(A B)$ $R_2(A C)$

② $R_1 \cup R_2 = R$

$$AB \cup AC = ABC$$

③ $ABC = ABC$

④ $R_1 \cap R_2 \neq \emptyset$

$$AB \cap AC \neq \emptyset$$

$$A \neq \emptyset$$

⑤ $R_1 \cap R_2 = \emptyset$

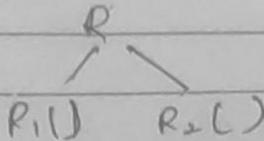
* Dependency Preserving Decomposition :-

$R(A B C D)$

FD $A \rightarrow B, B \rightarrow C$

~~$B \rightarrow C$~~ $A \rightarrow C$ (From transitive property)

FD*



Functional
Dependency PDI FD2

also get
derived

$$FD_1 \cup FD_2 = FD^*$$

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

$\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$

R is decomposed into $R_1(A B), R_2(C D), R_3(B D)$

$R_1(A B)$	$R_2(C D)$	$R_3(B D)$
$A \rightarrow A$	$B \rightarrow C$ ✓	$B \rightarrow D$ ✓
$B \rightarrow B$	$C \rightarrow B$ ✓	$D \rightarrow B$ ✓
$A \rightarrow B$ ✓	$C \leftarrow C B$	
$B \rightarrow A$ X		
$B^+ = B C D$		

✓ $A \rightarrow B$

✓ $B \rightarrow C$

$C \rightarrow B$

$B \rightarrow D$

$D \rightarrow B$

Now, we will check that whether we can derive $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$

from this F.D.

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

~~$C \rightarrow B$~~

~~$C \rightarrow B$~~

∴ all 4 F.D are
preserved.

$$C^+ = C B D$$

Fourth Normal Form (4NF)

A relation 'R' is in 4NF if and only if the following condition are satisfied -

- i) if 'R' is in 3NF or BCNF
- ii) if it contains no MVD's

Now What is Multivalued Dependency
For a dependency $x \rightarrow y$, if for a single value of x , multiple value of y exists, then the relation have multivalued dependency.

- The relation should have at least three attributes $(x \rightarrow y), (x \rightarrow z)$
- The attributes y and z should be independent of each other.

Note [F.D $(\alpha \rightarrow \beta)$ says we can't have two tuples with same α value but different β value].

Consider multivalued dependency

E_id	Project Name	D_Name
101	Java/DBMS	Sita/Khushi
102	OS/CN	Geeba/Ram

$E_id \rightarrow\rightarrow Project\ Name$

$E_id \rightarrow\rightarrow D_Name$

employee-id "multidetermines" D_Name

In this ex - Project Name and
D_Name both are independent
attributes and it is dependent
on e_id.

Ex Consider the database table
of a class which has two
relations R1 contains studentID(SID)
and student name (SNAME) and
R2 contains courseID (CID) and
course name (CNAME)

R1 (SID, SNAME)

SID	SNAME
S1	A
S2	B

R2 (CID, CNAME)

CID	CNAME
C1	C
C2	D

Table R1XR2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Multivalued dependencies (MVD) are
 $SID \rightarrow\!\!> CID$; $SID \rightarrow\!\!> CNAME$;
 $SNAME \rightarrow\!\!> CNAME$

Note \rightarrow MVD occurs if two or more independent relations are kept in a single relation.

Note - UNF is a level of a database normalization where there are no non-trivial multivalued dependencies other than a candidate key.

How to decompose it in 4NF?

E-id	Project Name	D-Name
101	Java/DBMS	Sita / Khushi
102	OS/CN	Gaurav / Ram

4

Eid	Project Name	D-Name
101	Java	Sita
101	DBMS	Sita
101	Java	Khushi
101	DBMS	Khushi
102	OS	Gaurav
102	CN	Gaurav
102	OS	Ram
102	CN	Ram

R₁

4

Eid	Project Name
101	Java
101	DBMS
102	OS
102	CN

Eid	D-Name
102	Sita
101	Khushi
102	Gaurav
102	Ram

5th NF or Project Join Normal Form

- 5NF is rarely used practically but it is useful for theoretical study.
- 5NF is based on Join dependency.
- Join dependency
 - Decompose the relation in multiple relations and it should be lossless and maintain dependencies of original relation.
- A relation is in 5NF -
 - i) It must be in 4NF
 - ii) No Join Dependency exists

For ex →

Dept	subject	student
CSE	DBMS	Shreya
IT	CN	Yug
CSE	OS	Geeta
CSE	COA	Sita
ME	APP	Rini
EC	CSA	Pra

Dept → → subject

Dept → → student

Dept	subject
CSE	DBMS
IT	CN
CSE	OS
CSE	COA
ME	APP
EC	CSA

Dept	student
CSE	Shreya
IT	Yug
CSE	Geeta
CSE	Sita
ME	Rini
EC	Pra

If we perform join here

Dept	subject	student
CSE	DBMS	Shreya ✓
CSE	DBMS	Geeta ✗
CSE	DBMS	Sita ✗
IT	CN	Yug ✓
CSE	OS	Shreya
CSE	OS	Geeta ✓
CSE	OS	Sita ✗
CSE	COA	Shreya ✗
CSE	COA	Geeta ✗
CSE	COA	Sita ✓
ME	APP	Rini ✓
EC	CSE	Pea ✓

so we have to decompose it
in 3 Table

Dept	subject

Dept	student

Subject	Student