DATABASE MANAGEMENT SYSTEMS

Authors – Raghu Ramakrishna, Johannes Gehrke

Edition – 3

UNIT-III

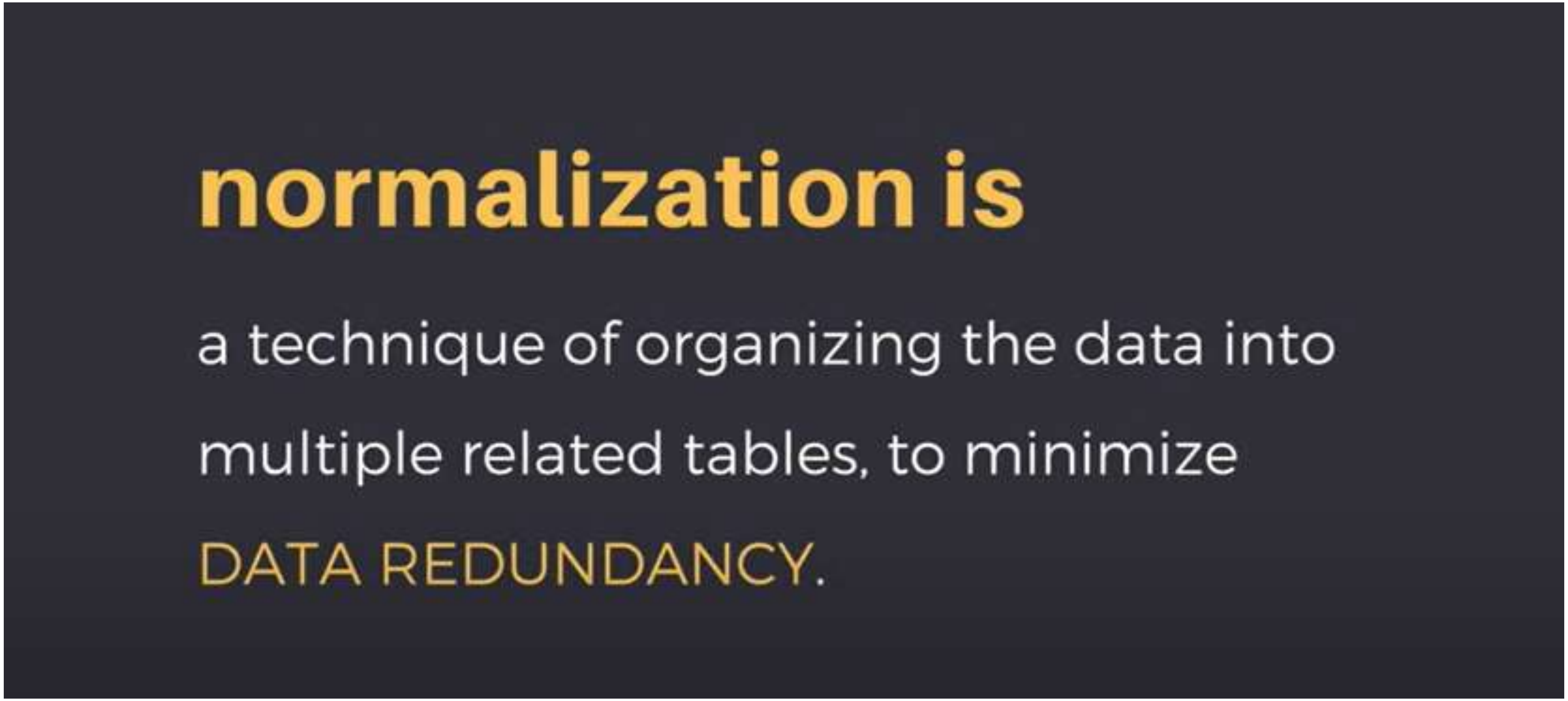
Schema refinement – Problems Caused by redundancy – Decompositions – Problem related to decomposition – reasoning about FDS – FIRST, SECOND, THIRD Normal forms – BCNF– Schema refinement in Database Design – Multi valued Dependencies – FOURTH Normal Form.

Schema Refinement

- The ***Schema Refinement*** refers to refine the schema by using some technique. The best technique of schema refinement is decomposition.
- Normalization or Schema Refinement is a technique of organizing the data in the database.
- It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like ***Insertion, Update and Deletion Anomalies***.
- **Redundancy:** refers to repetition of same data or duplicate copies of same data stored in different locations.
- **Anomalies**: refers to the problems occurred after poorly planned and normalized databases where all the data is stored in one table which is sometimes called a **flat file database**.

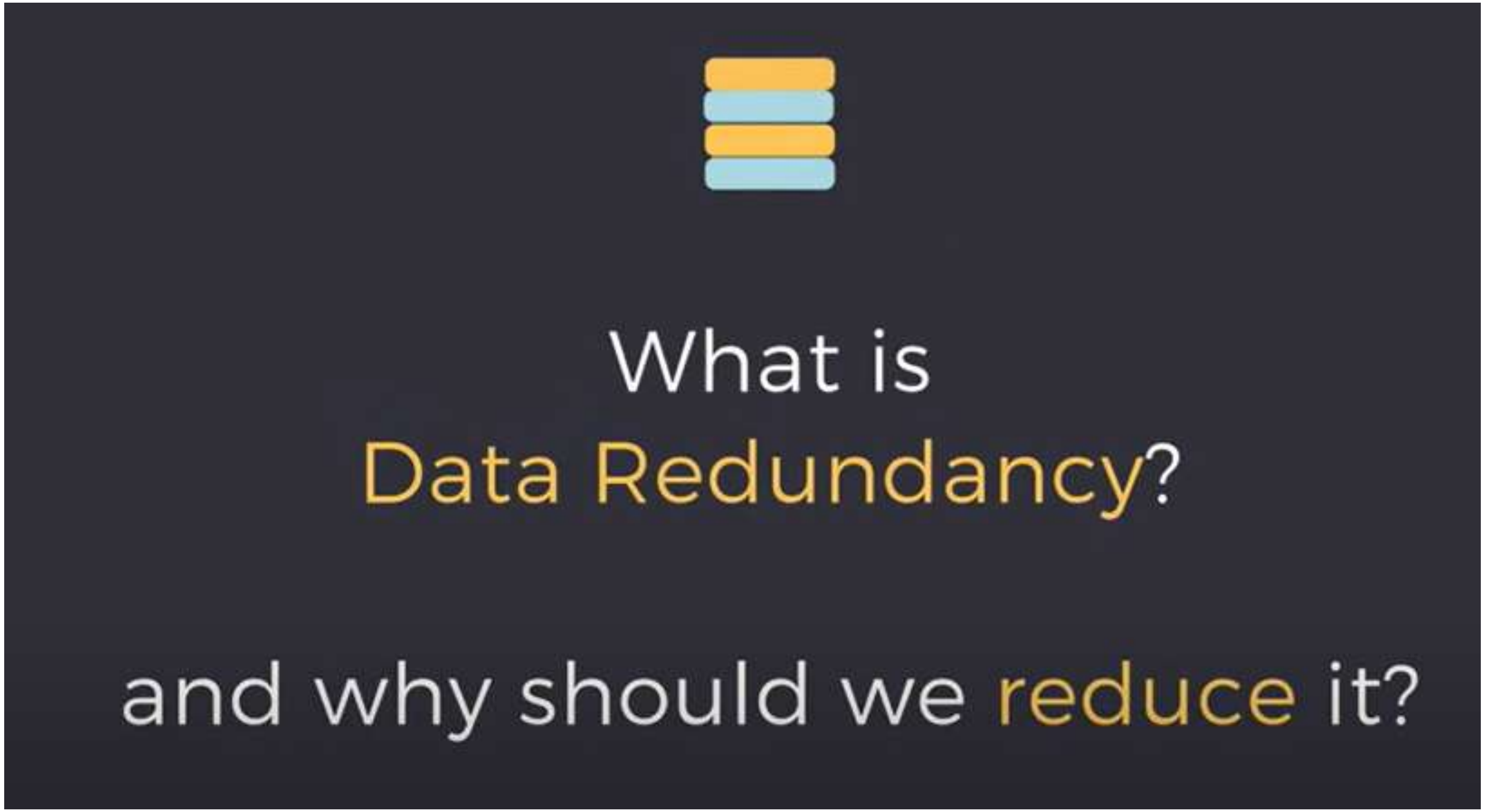
Database

NORMALIZATION



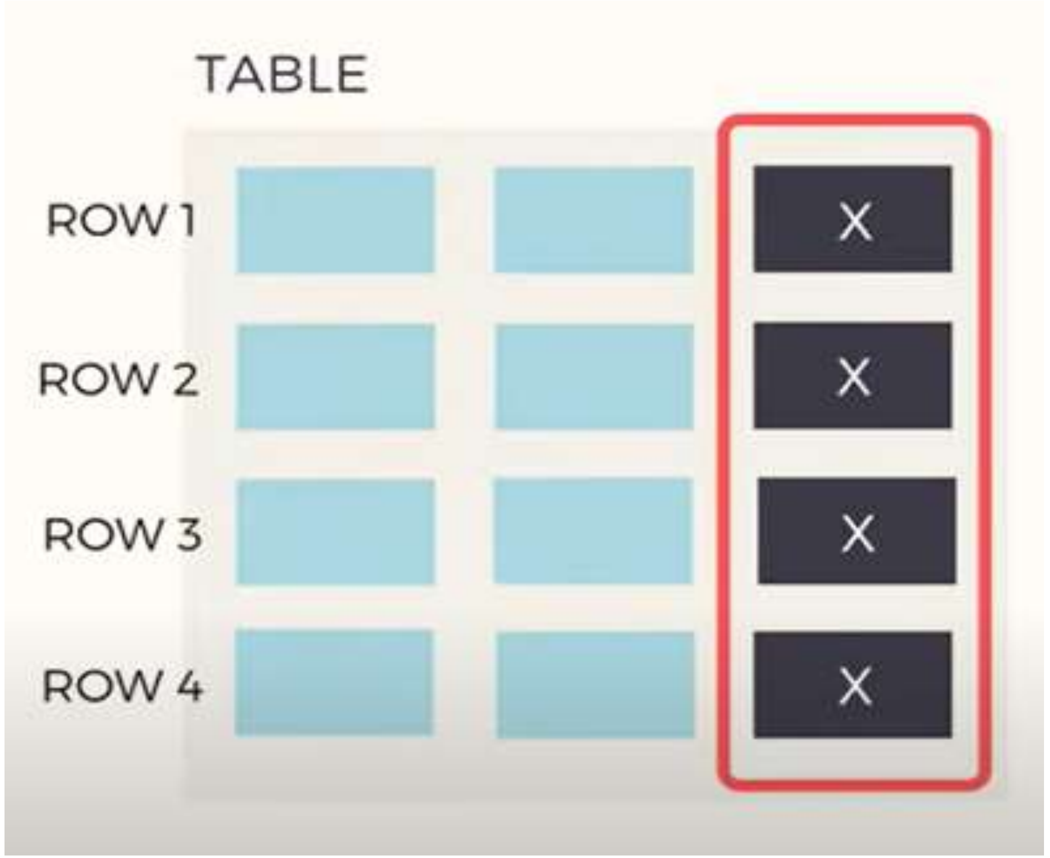
normalization is

a technique of organizing the data into multiple related tables, to minimize DATA REDUNDANCY.



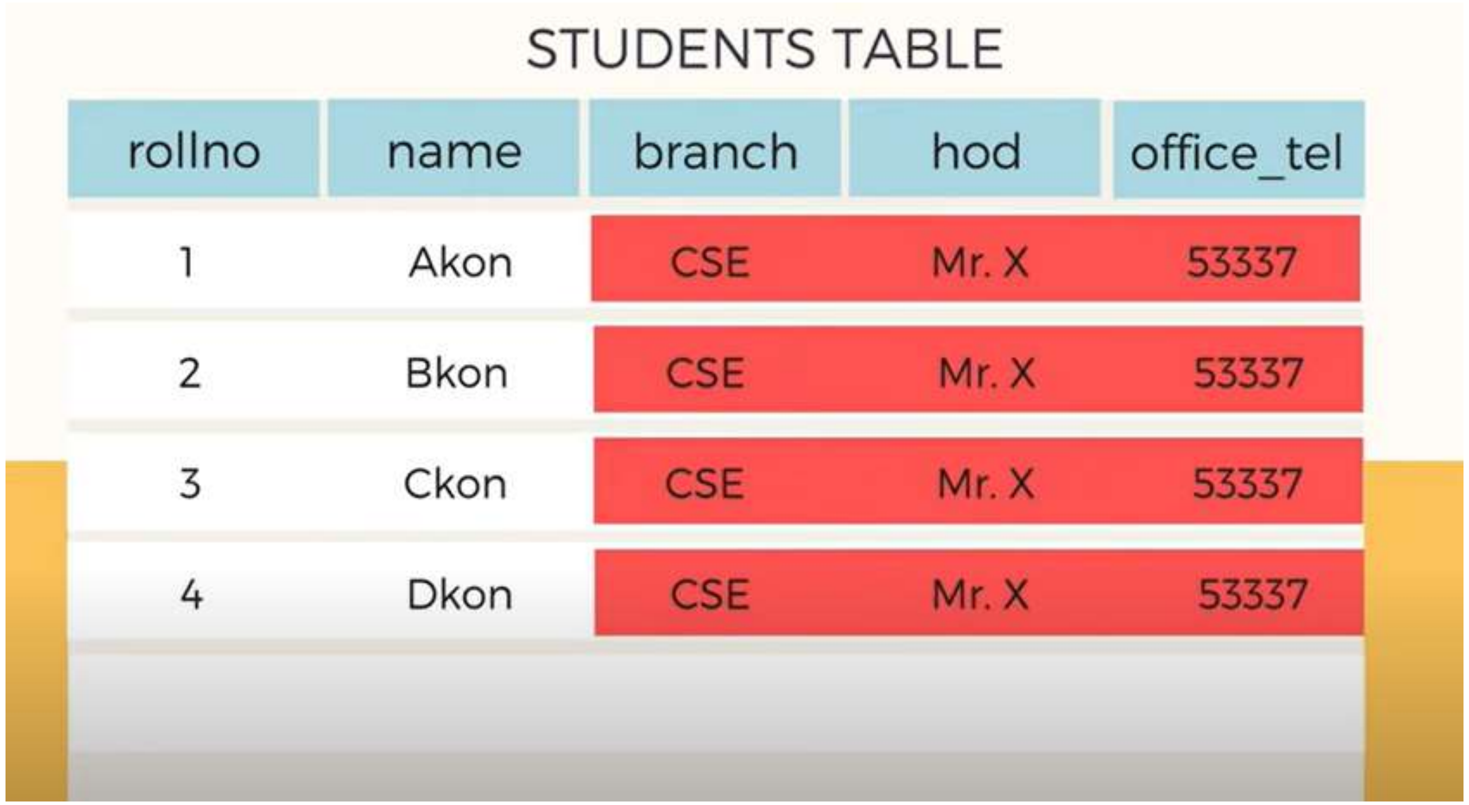
What is
Data Redundancy?

and why should we reduce it?



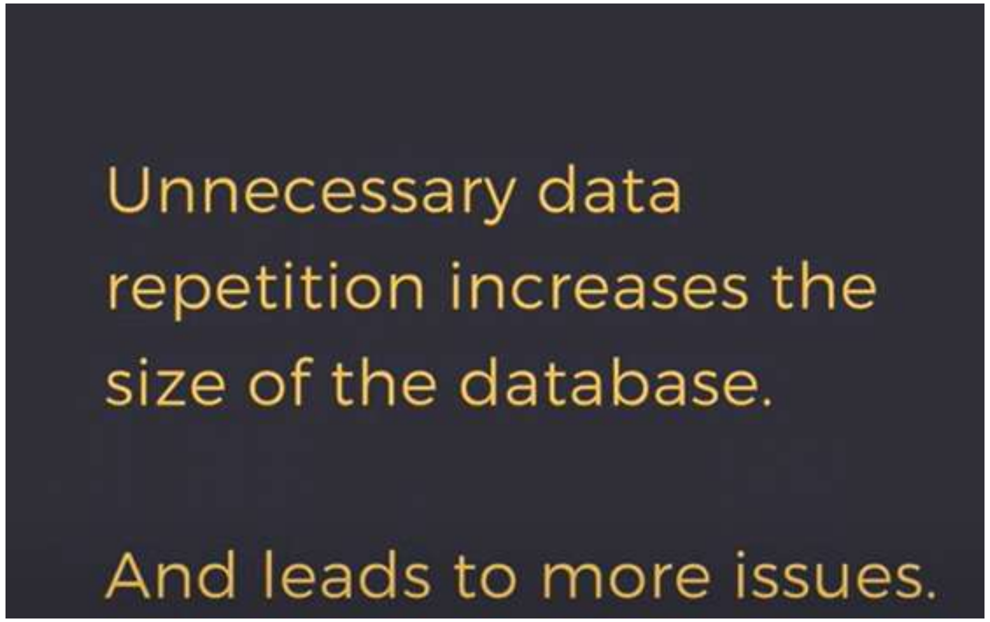
- Repetition of data increases the size of database.
- Other issues like:
  - Insertion Problems
  - Deletion Problems
  - Updation Problems

EXAMPLE:



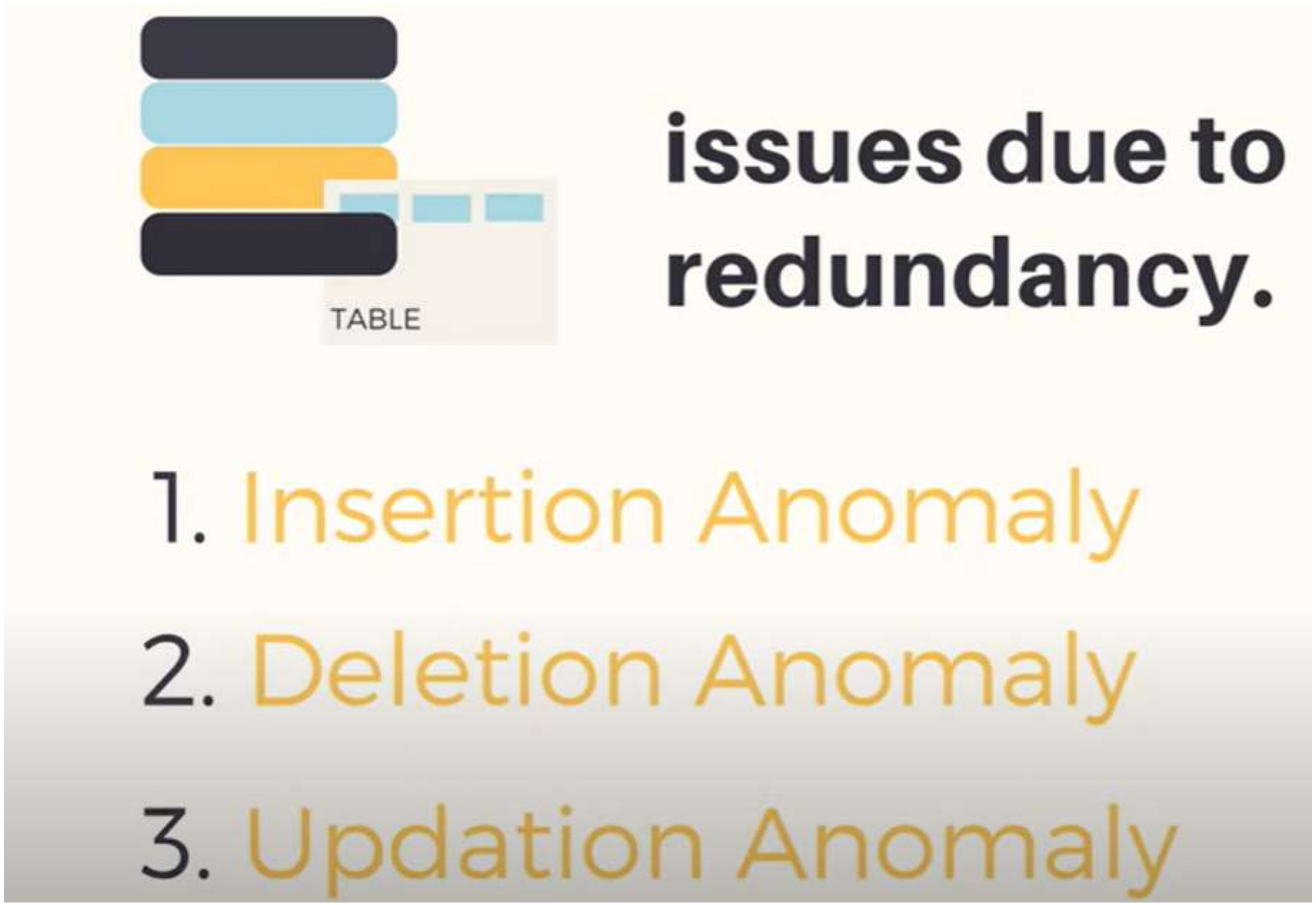
STUDENTS TABLE

rollno	name	branch	hod	office_tel
1	Akon	CSE	Mr. X	53337
2	Bkon	CSE	Mr. X	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	Mr. X	53337



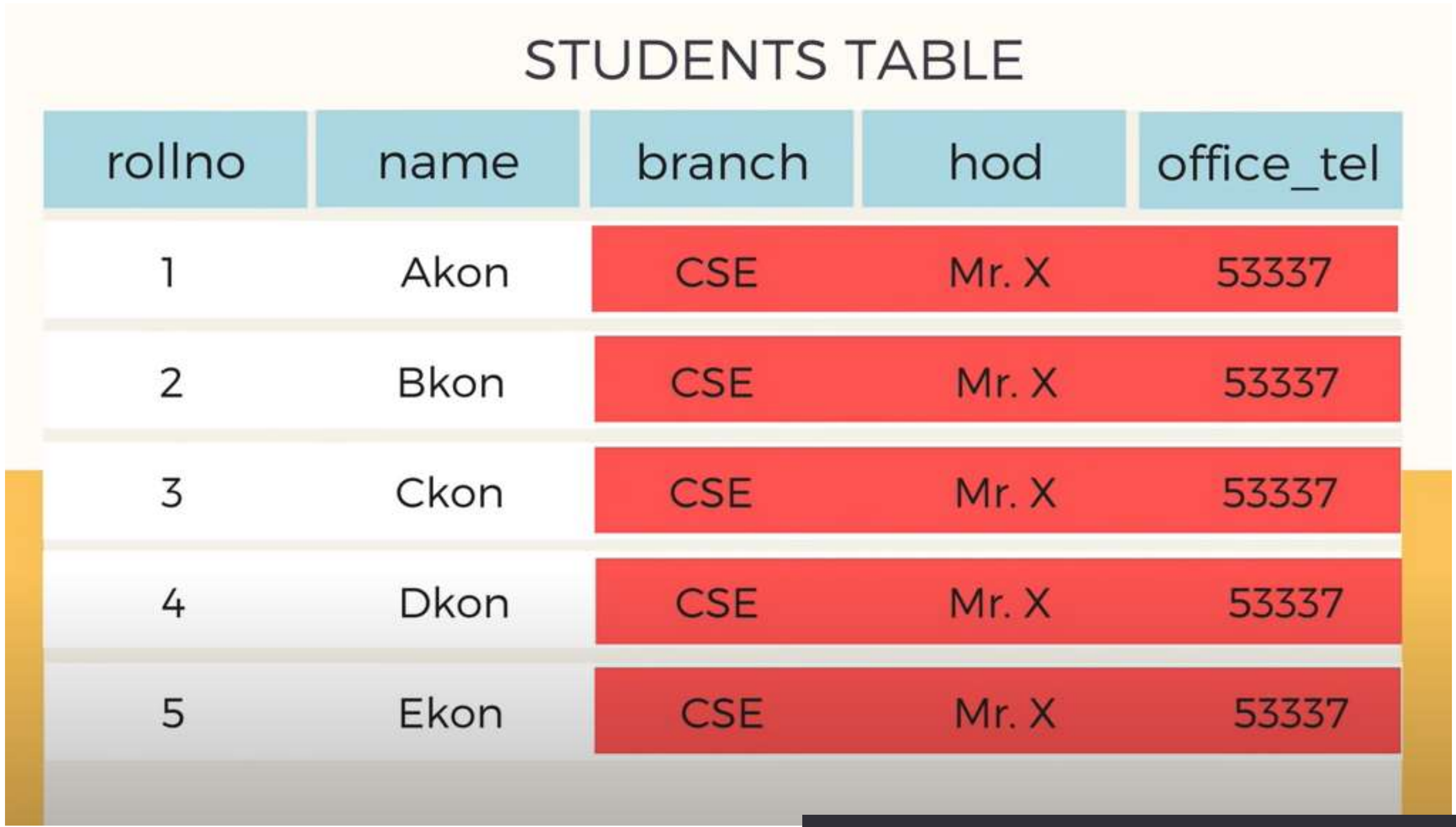
Unnecessary data repetition increases the size of the database.

And leads to more issues.



issues due to redundancy.

1. Insertion Anomaly
2. Deletion Anomaly
3. Updation Anomaly



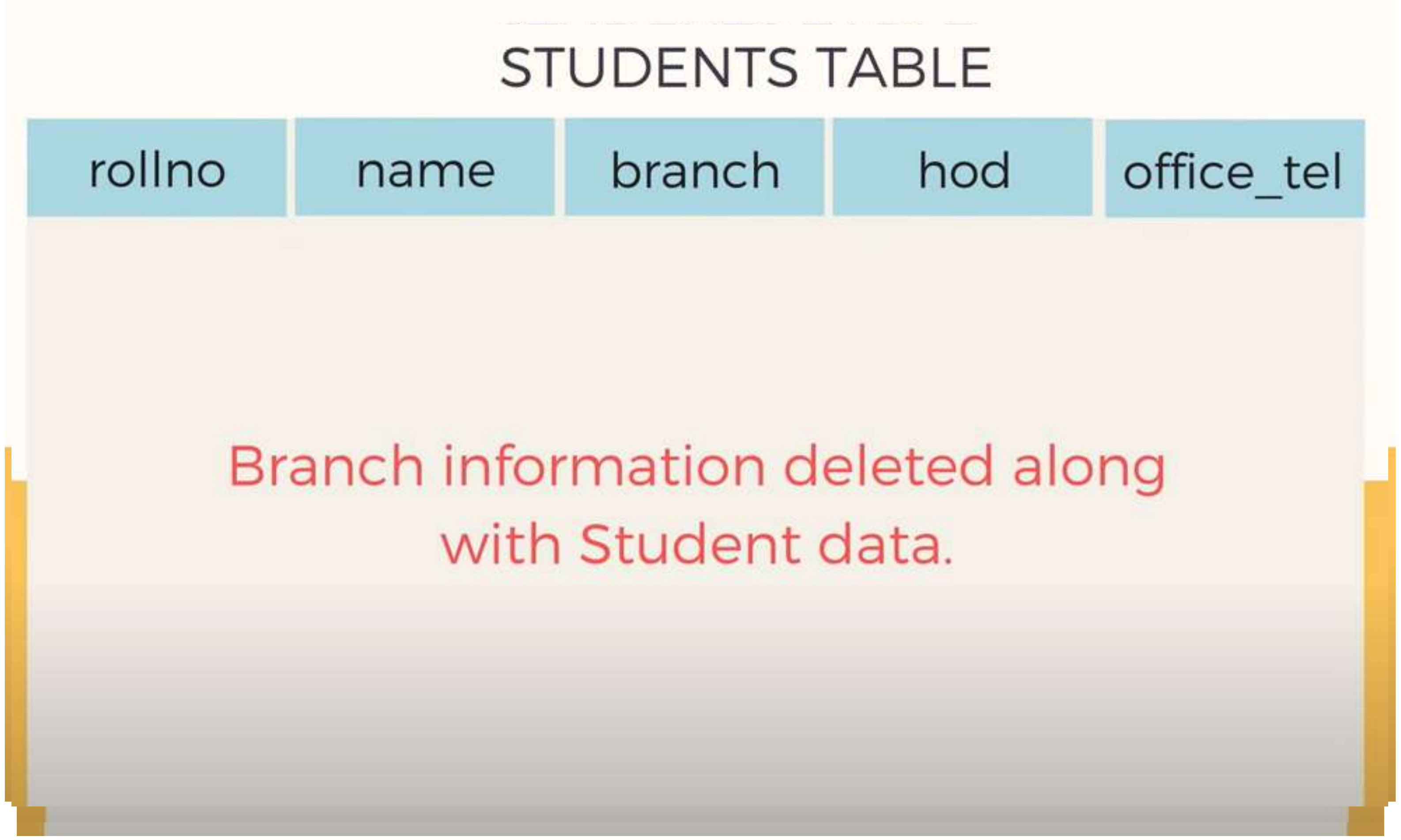
STUDENTS TABLE

rollno	name	branch	hod	office_tel
1	Akon	CSE	Mr. X	53337
2	Bkon	CSE	Mr. X	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	Mr. X	53337
5	Ekon	CSE	Mr. X	53337

Insertion Anomaly

To insert redundant data for every new row (of Student data in our case) is a data insertion problem or anomaly.

Deletion Anomaly



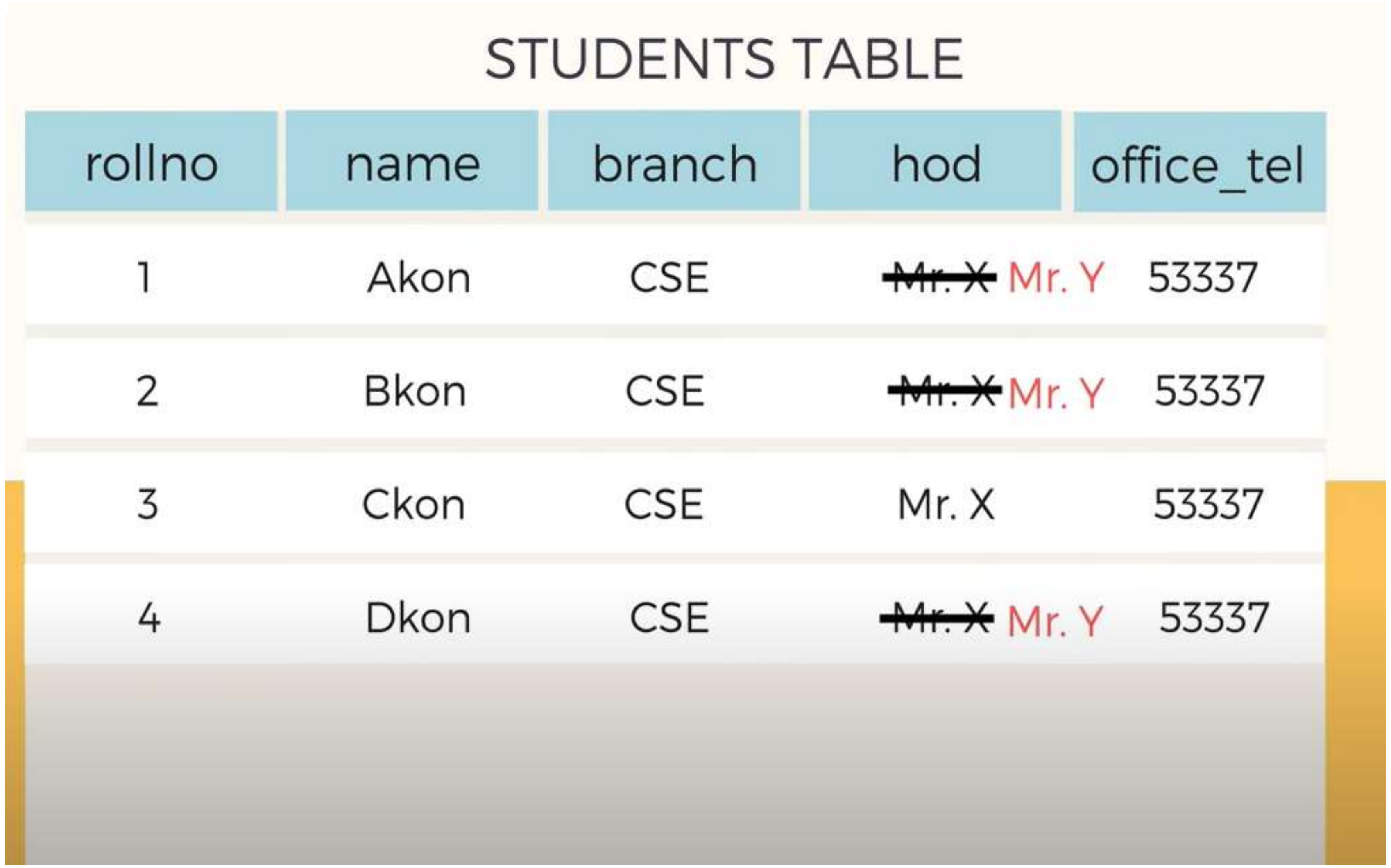
STUDENTS TABLE

rollno	name	branch	hod	office_tel

Branch information deleted along with Student data.

Loss of a related dataset when some other dataset is deleted.

Updating Anomaly



STUDENTS TABLE

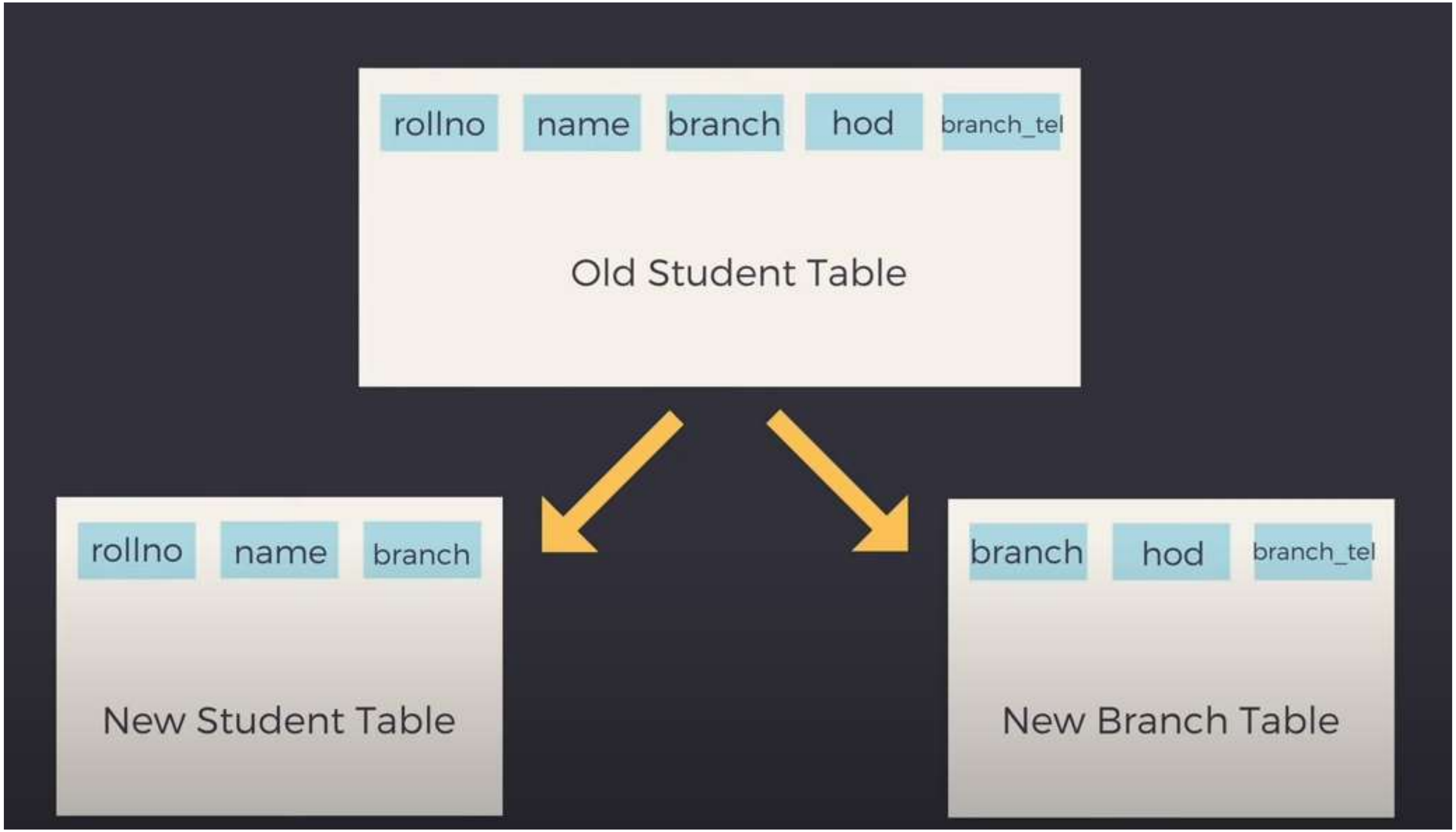
rollno	name	branch	hod	office_tel
1	Akon	CSE	~~Mr. X~~ Mr. Y	53337
2	Bkon	CSE	~~Mr. X~~ Mr. Y	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	~~Mr. X~~ Mr. Y	53337

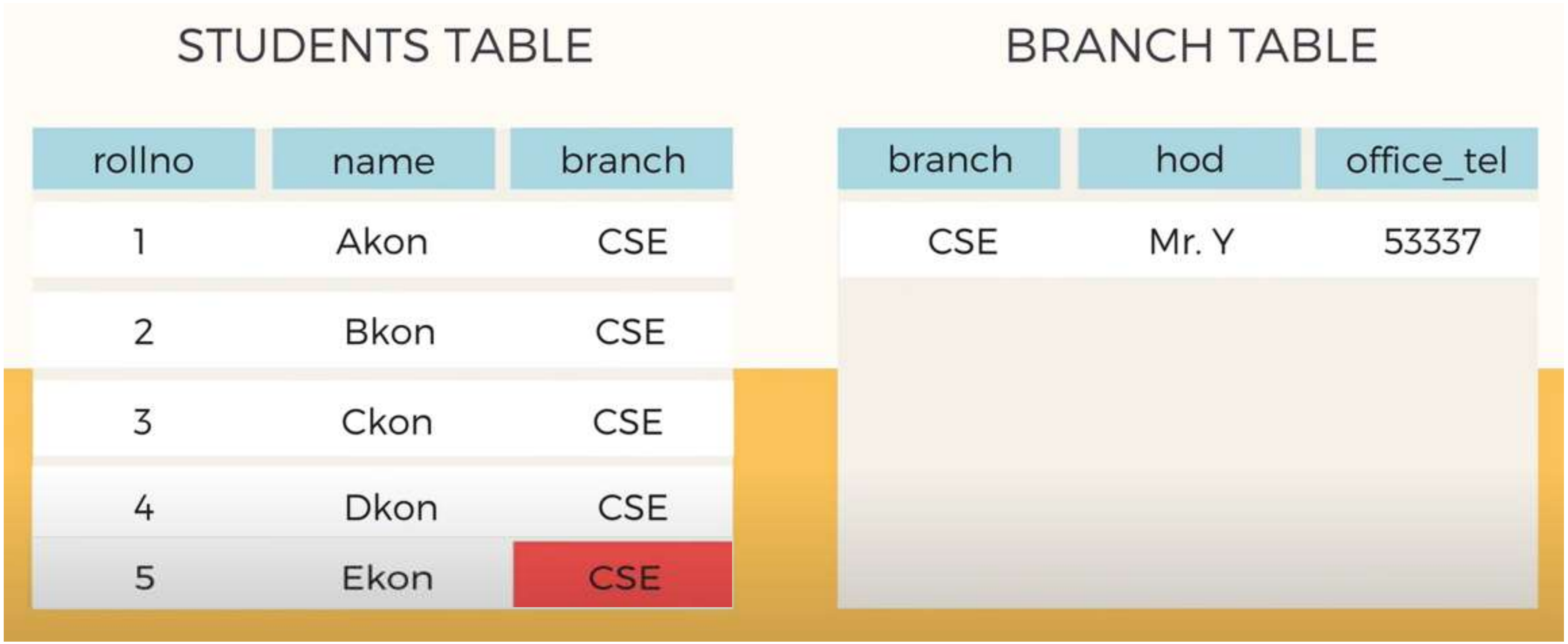
How Normalization will solve all these problems?

Student Table

$\downarrow$

Student Table + Branch Table


rollno
name
branch
hod
branch_tel
Old Student Table
rollno
name
branch
New Student Table
branch
hod
branch_tel
New Branch Table



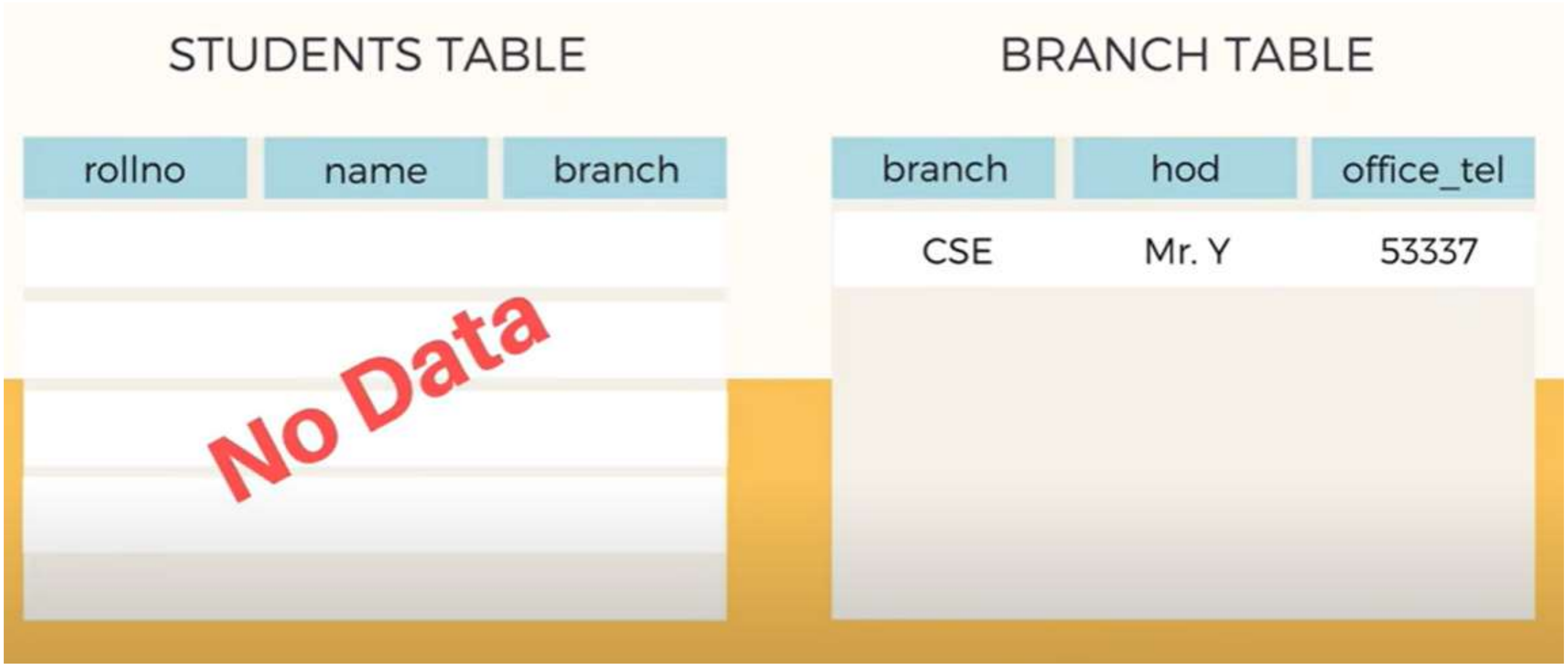
STUDENTS TABLE

rollno	name	branch
1	Akon	CSE
2	Bkon	CSE
3	Ckon	CSE
4	Dkon	CSE
5	Ekon	CSE

BRANCH TABLE

branch	hod	office_tel
CSE	Mr. Y	53337

Insertion problem solved



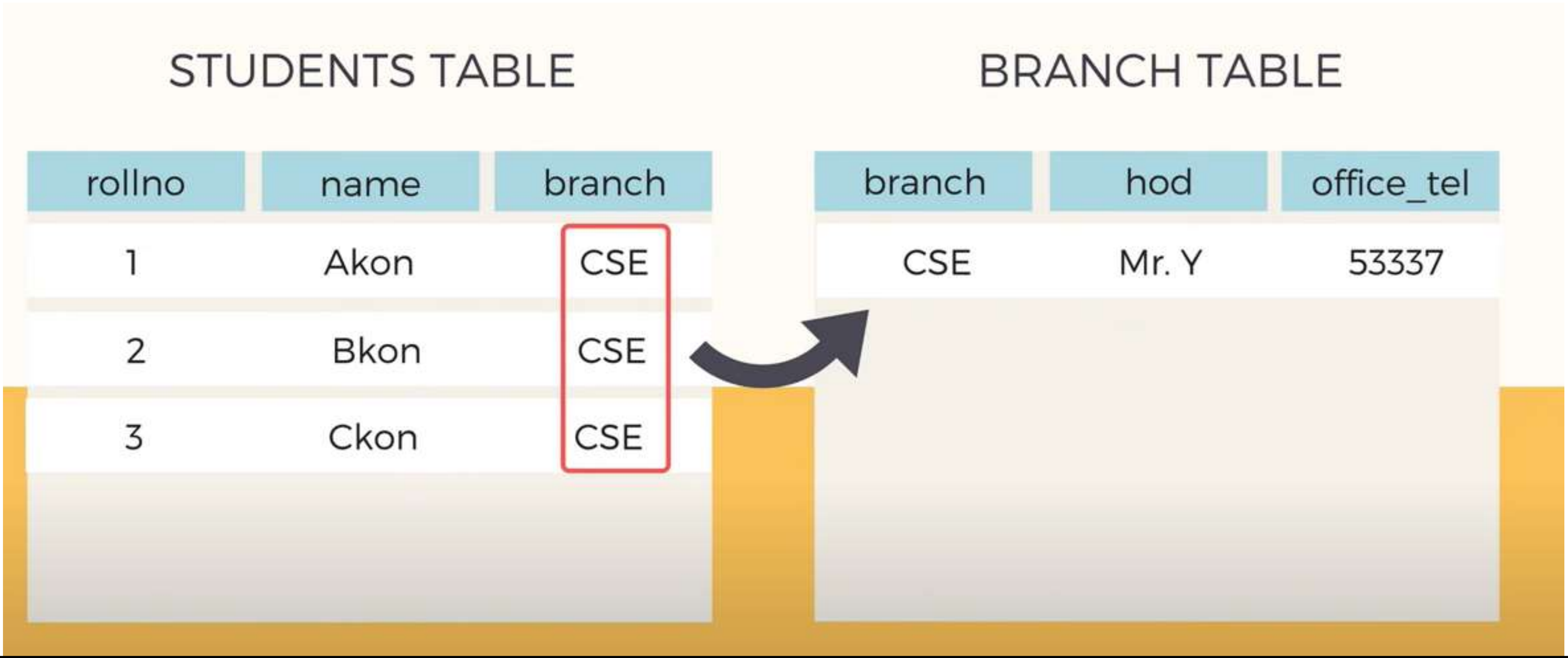
STUDENTS TABLE

rollno	name	branch

BRANCH TABLE

branch	hod	office_tel
CSE	Mr. Y	53337

Deletion problem solved



STUDENTS TABLE

rollno	name	branch
1	Akon	CSE
2	Bkon	CSE
3	Ckon	CSE

BRANCH TABLE

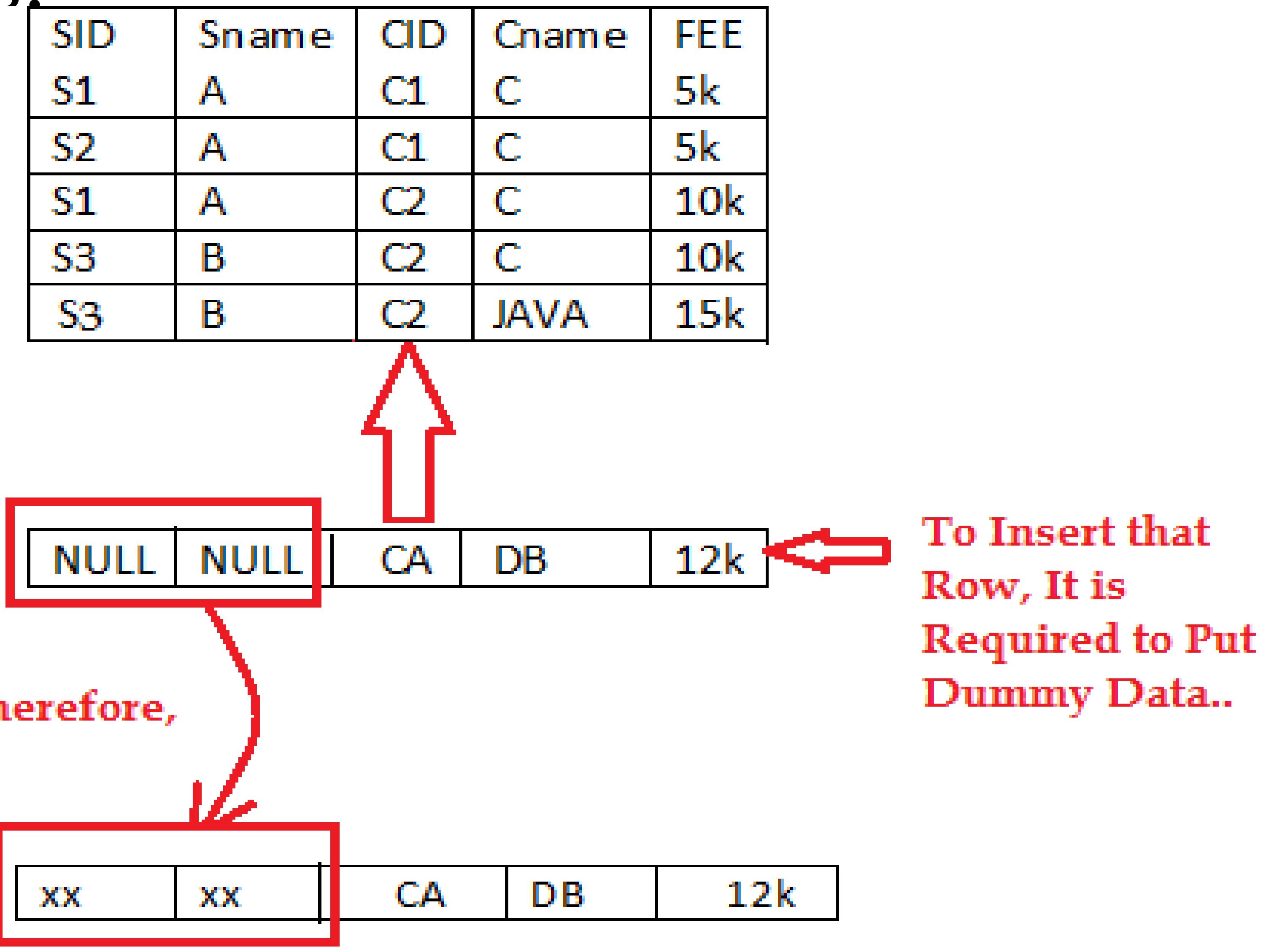
branch	hod	office_tel
CSE	Mr. Y	53337

Whenever we update branch table automatically reflect to the students table. This is called Minimization of Redundancy but not eliminating the Redundancy

Anomalies or problems facing without normalization (problems caused by redundancy) :

- ***Redundant Storage***: Repeatedly storing information.
- ***Update Anomalies***: If one copy of repeated data is updated, inconsistency is created unless all copies of data are updated.
- ***Insertion Anomalies***: It is not possible to insert certain information until some other waste information is stored with it.
- ***Deletion Anomalies***: It is not possible to delete certain information until some other useful information is deleted with it.

Insertion Anomalies:



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

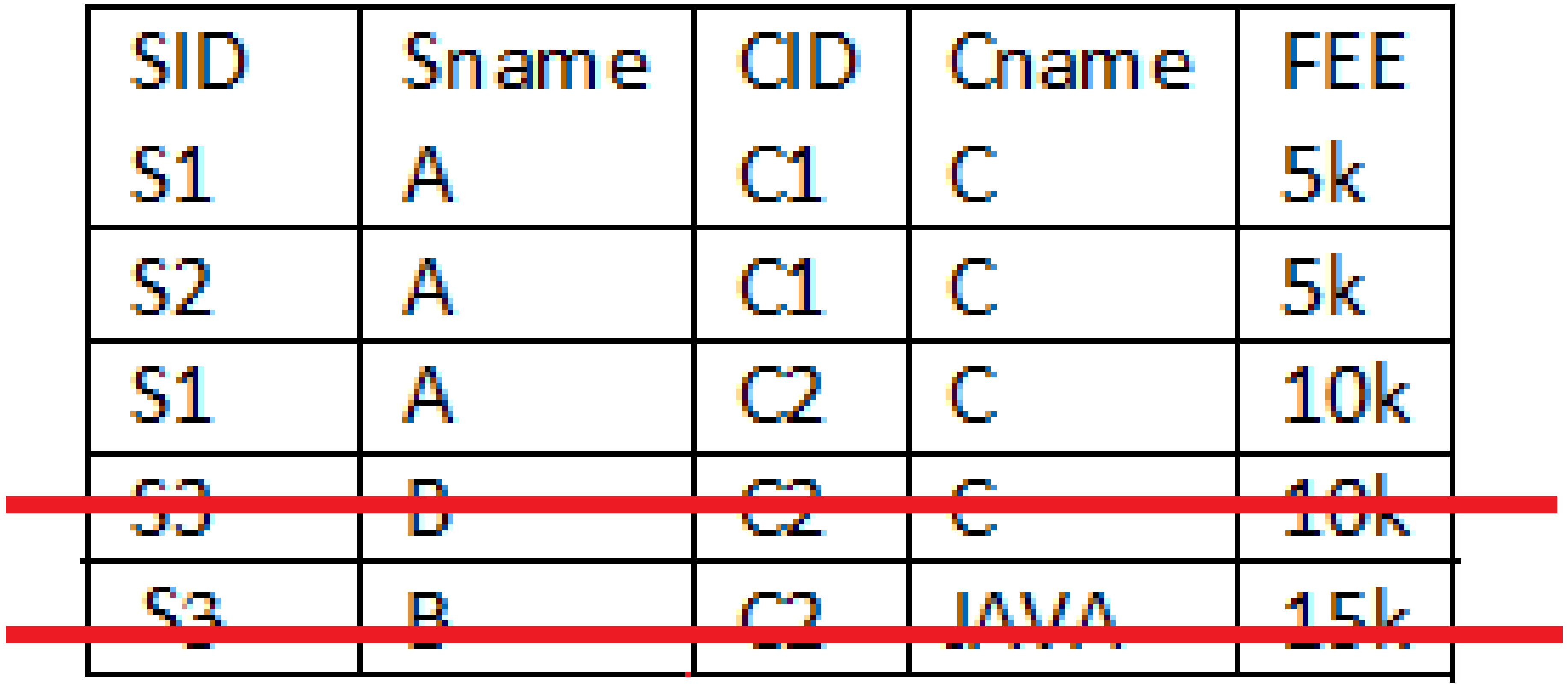
NULL	NULL	CA	DB	12k

To Insert that Row, It is Required to Put Dummy Data..

Therefore,

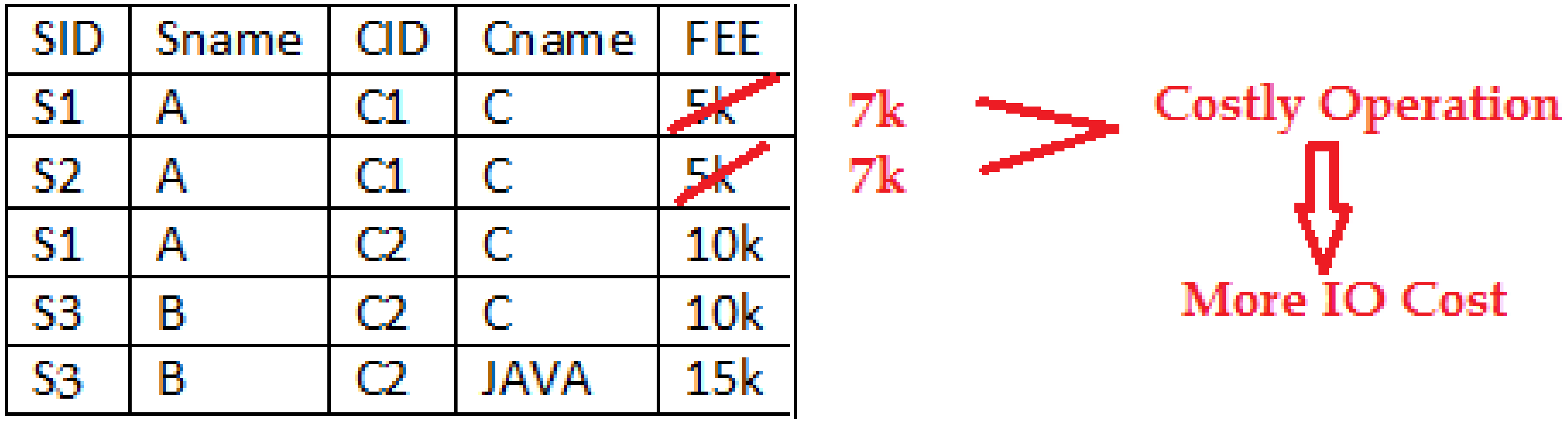
xx	xx	CA	DB	12k

Deletion Anomalies:



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

Update Anomalies:



SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~
S2	A	C1	C	~~5k~~
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

7k

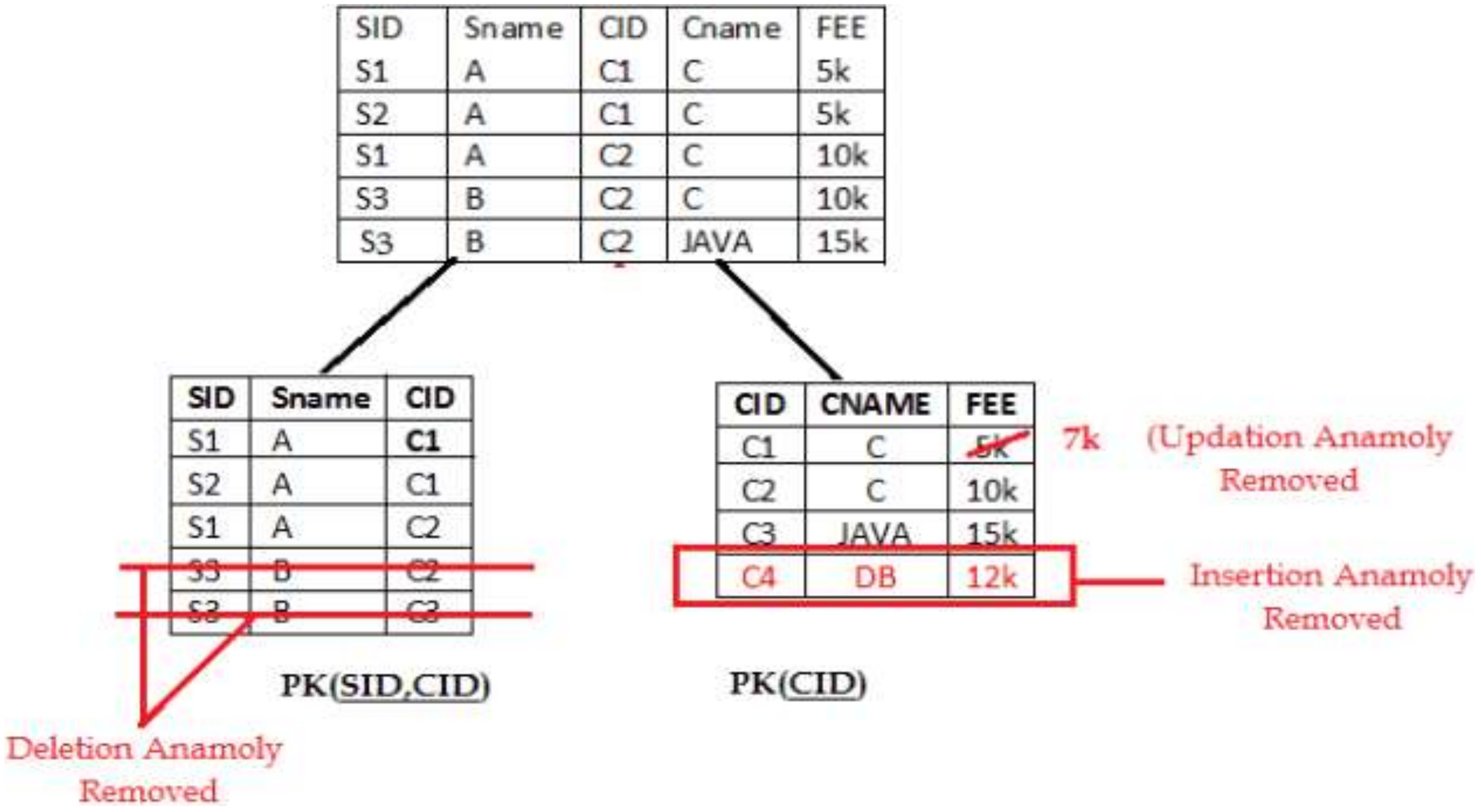
7k

Costly Operation

More IO Cost

Solutions To Anomalies :

Decomposition of Tables – Schema Refinement



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
~~S3~~	~~B~~	~~C2~~
~~S3~~	~~B~~	~~C3~~

PK(SID,CID)

CID	CNAME	FEE
C1	C	~~5k~~ 7k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

There are some Anomalies in this again –

SID	Sname	CID
S1	~~A~~ (AA)	C1
S2	~~A~~ (AA)	C1
S1	~~A~~ (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Updation Anamoly

A student having no course is enrolled. We have to put dummy data again.

Deletion Anamoly as C2 course is alloted to some students

CID	CNAME	FEE
C1	C	5k
~~C2~~	~~C~~	~~10k~~
C3	JAVA	15k
C4	DB	12k

What is the Solution ??

Solution : decomposing into relations as shown below

R1

SID	Sname

R2

SID	CID

R3

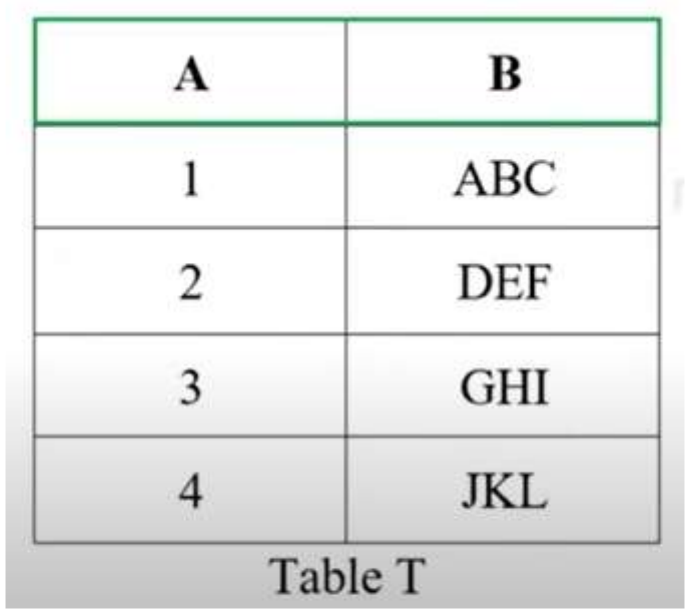
CID	Cname	Fee

Functional Dependency:

- **Functional** dependency is a **Constraint** that **determines** the relation of one attribute to another. It typically exists between the primary key and non-key attribute within a table.

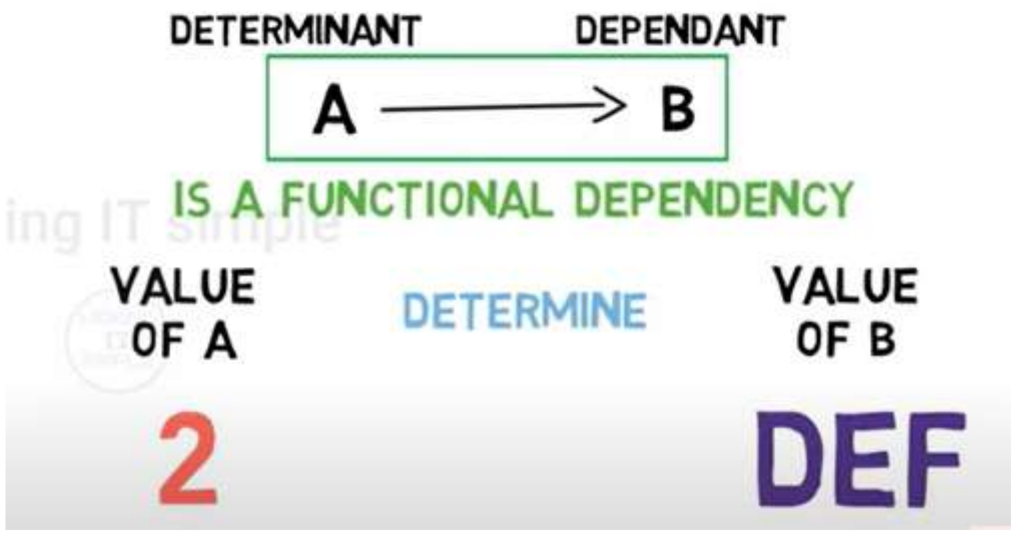
$$X \rightarrow Y$$

- The left side of FD is known as a ***Determinant***, the right side of the production is known as a ***Dependent*****.**

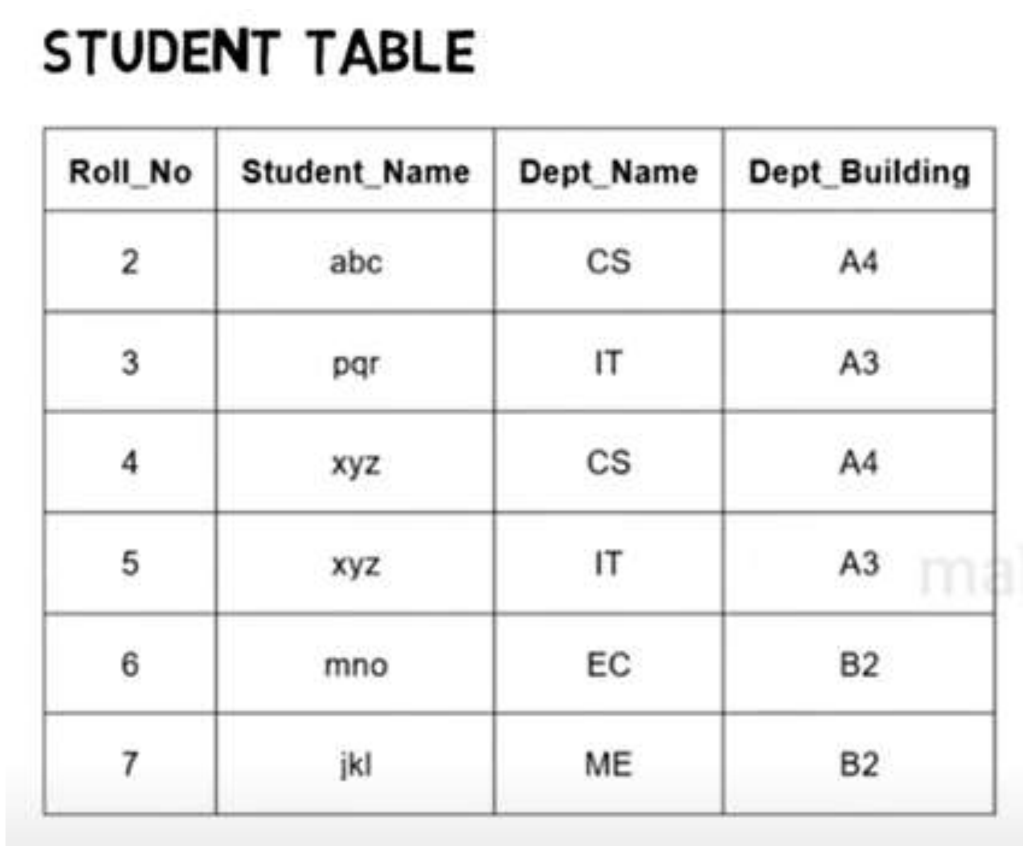


A	B
1	ABC
2	DEF
3	GHI
4	JKL

Table T

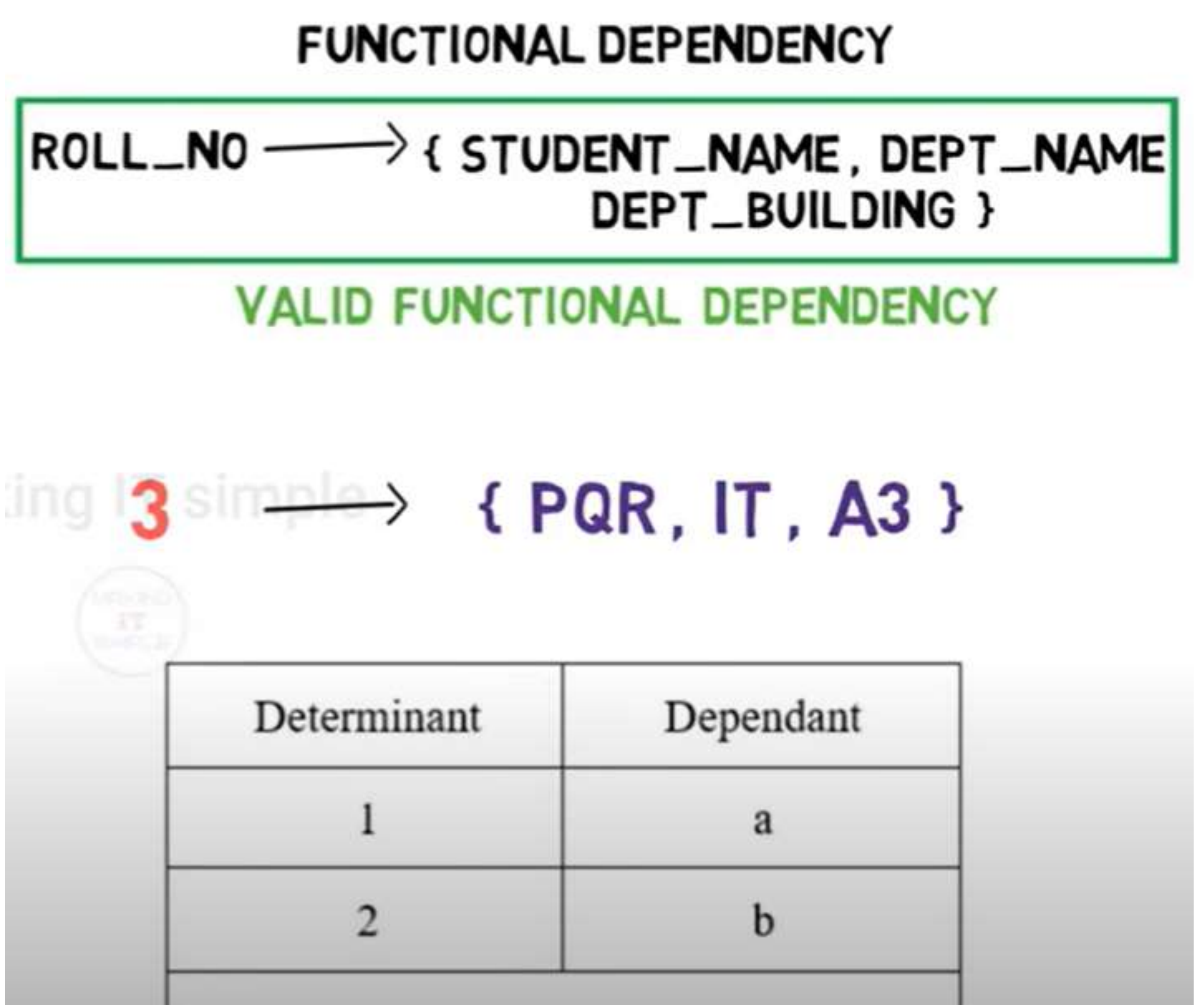


Scenario 1



STUDENT TABLE

Roll_No	Student_Name	Dept_Name	Dept_Building
2	abc	CS	A4
3	pqr	IT	A3
4	xyz	CS	A4
5	xyz	IT	A3
6	mno	EC	B2
7	jkl	ME	B2



FUNCTIONAL DEPENDENCY

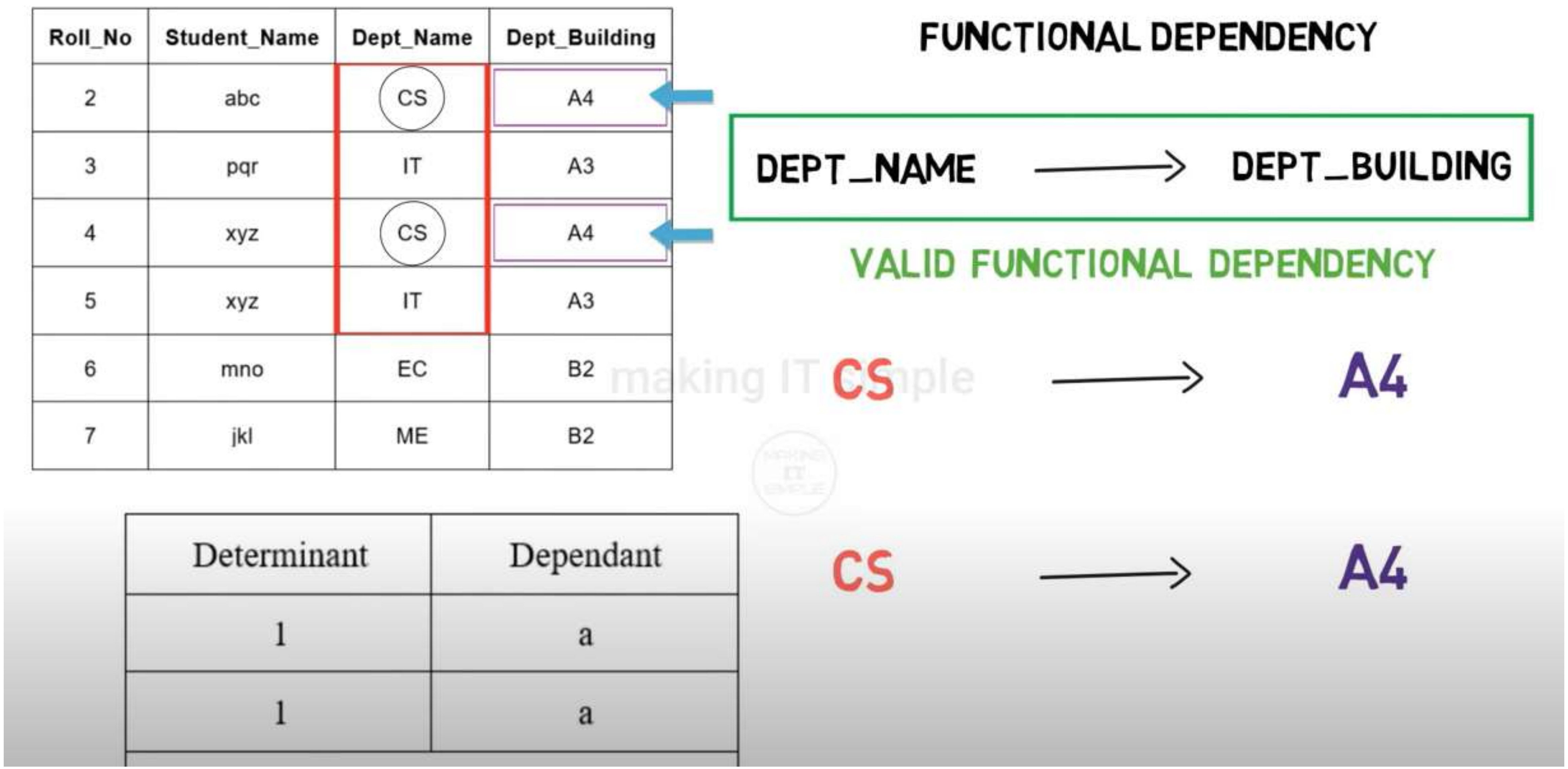
ROLL_NO ⟶ { STUDENT_NAME, DEPT_NAME DEPT_BUILDING }

VALID FUNCTIONAL DEPENDENCY

3 ⟶ { PQR, IT, A3 }

Determinant	Dependant
1	a
2	b

Scenario 2



Roll_No	Student_Name	Dept_Name	Dept_Building
2	abc	CS	A4
3	pqr	IT	A3
4	xyz	CS	A4
5	xyz	IT	A3
6	mno	EC	B2
7	jkl	ME	B2

FUNCTIONAL DEPENDENCY

$$\boxed{\text{DEPT\_NAME} \longrightarrow \text{DEPT\_BUILDING}}$$

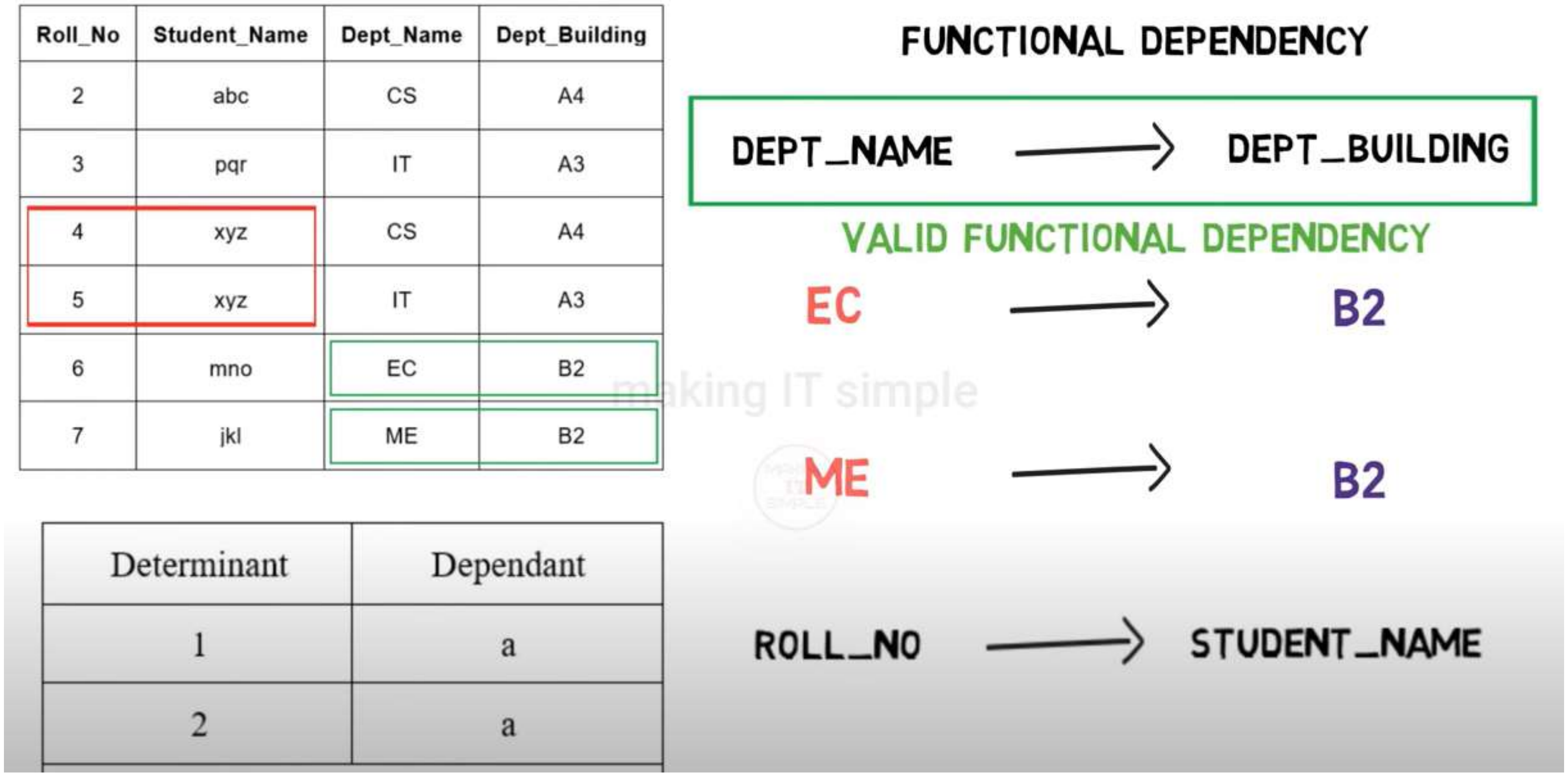
VALID FUNCTIONAL DEPENDENCY

$$\text{CS} \longrightarrow \text{A4}$$

$$\text{CS} \longrightarrow \text{A4}$$

Determinant	Dependant
1	a
1	a

Scenario 3



Roll_No	Student_Name	Dept_Name	Dept_Building
2	abc	CS	A4
3	pqr	IT	A3
4	xyz	CS	A4
5	xyz	IT	A3
6	mno	EC	B2
7	jkl	ME	B2

FUNCTIONAL DEPENDENCY

$\boxed{\text{DEPT\_NAME} \longrightarrow \text{DEPT\_BUILDING}}$

VALID FUNCTIONAL DEPENDENCY

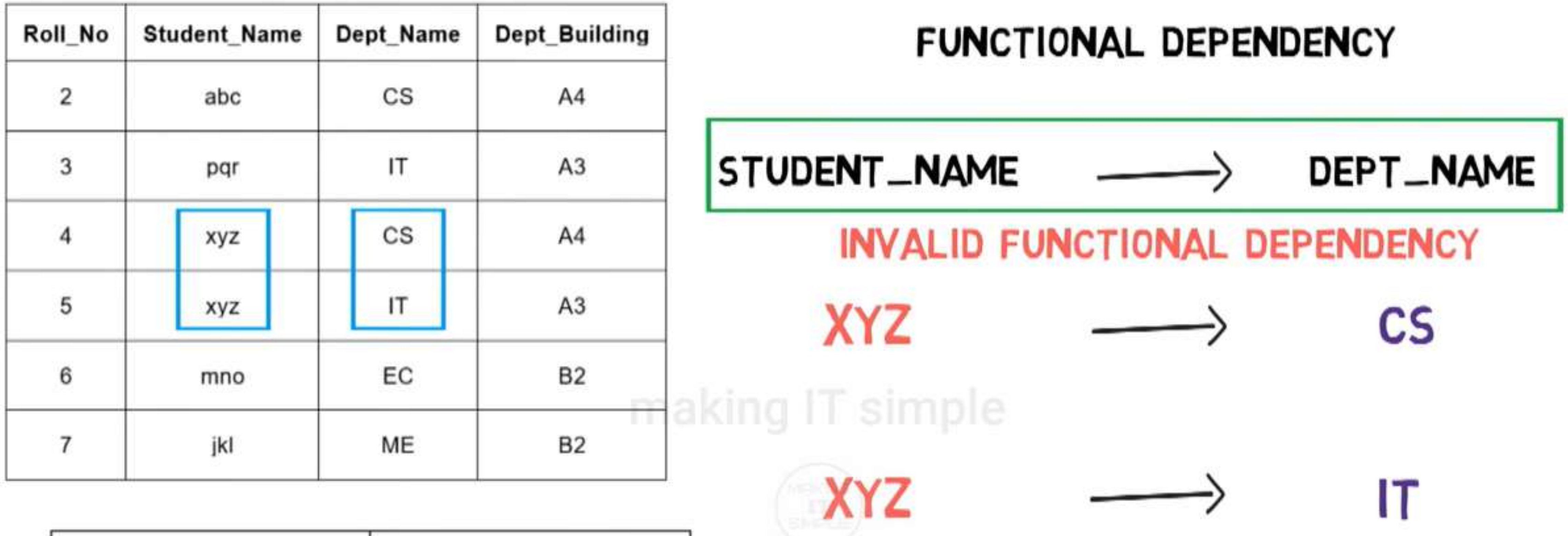
EC $\longrightarrow$ B2

ME $\longrightarrow$ B2

Determinant	Dependant
1	a
2	a

ROLL_NO $\longrightarrow$ STUDENT_NAME

Scenario 4



Roll_No	Student_Name	Dept_Name	Dept_Building
2	abc	CS	A4
3	pqr	IT	A3
4	xyz	CS	A4
5	xyz	IT	A3
6	mno	EC	B2
7	jkl	ME	B2

FUNCTIONAL DEPENDENCY

$$\boxed{\text{STUDENT\_NAME} \longrightarrow \text{DEPT\_NAME}}$$

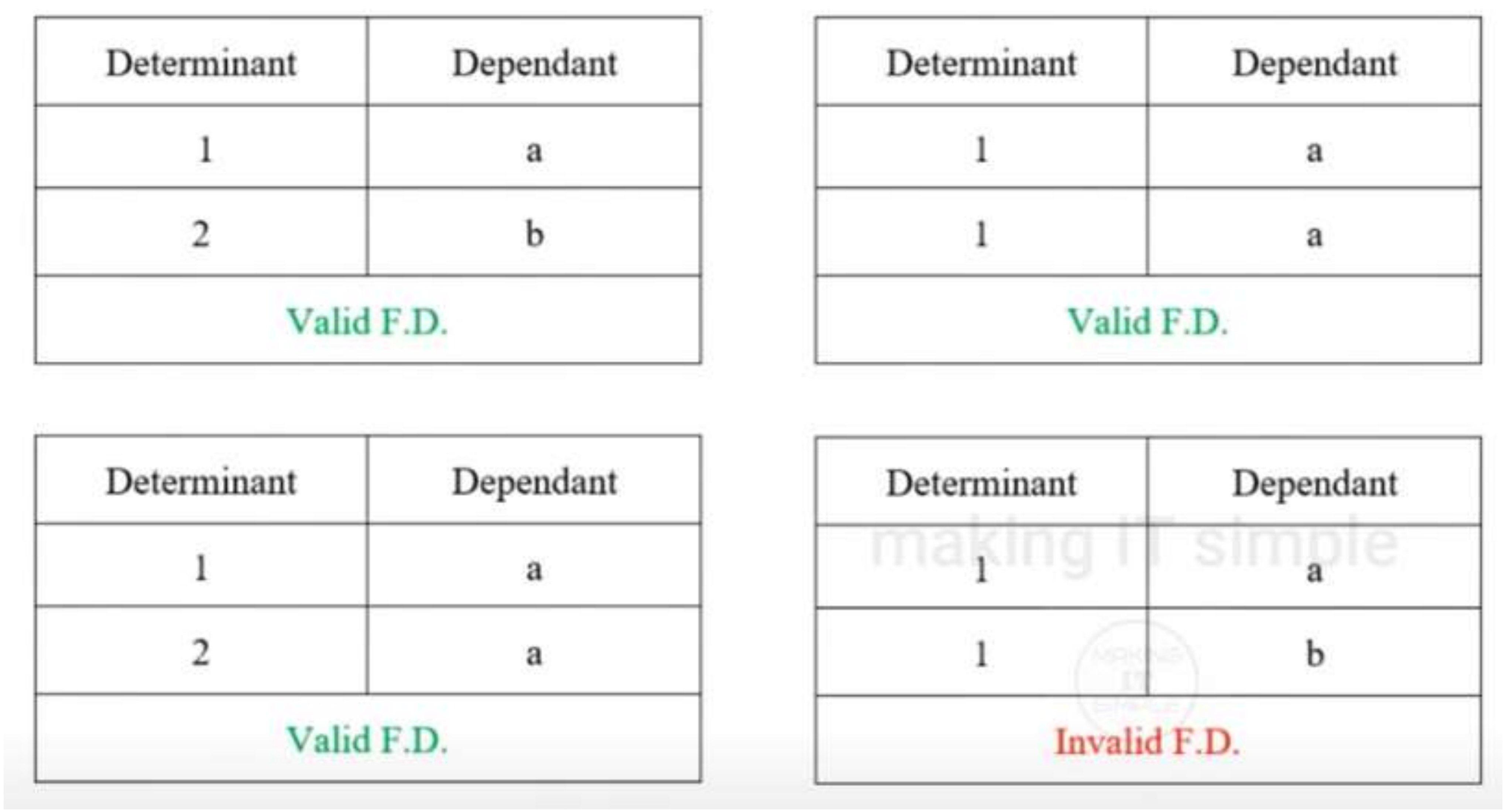
INVALID FUNCTIONAL DEPENDENCY

$$\text{XYZ} \longrightarrow \text{CS}$$

$$\text{XYZ} \longrightarrow \text{IT}$$

Determinant	Dependant
1	a
1	b

Functional Dependency:



Determinant	Dependant
1	a
2	b
Valid F.D.	

Determinant	Dependant
1	a
1	a
Valid F.D.	

Determinant	Dependant
1	a
2	a
Valid F.D.	

Determinant	Dependant
1	a
1	b
Invalid F.D.	

TYPES OF FUNCTIONAL DEPENDENCY

FUNCTIONAL DEPENDENCY

PARTIAL DEPENDENCY

TRIVIAL FUNCTIONAL DEPENDENCY

NON TRIVIAL FUNCTIONAL DEPENDENCY

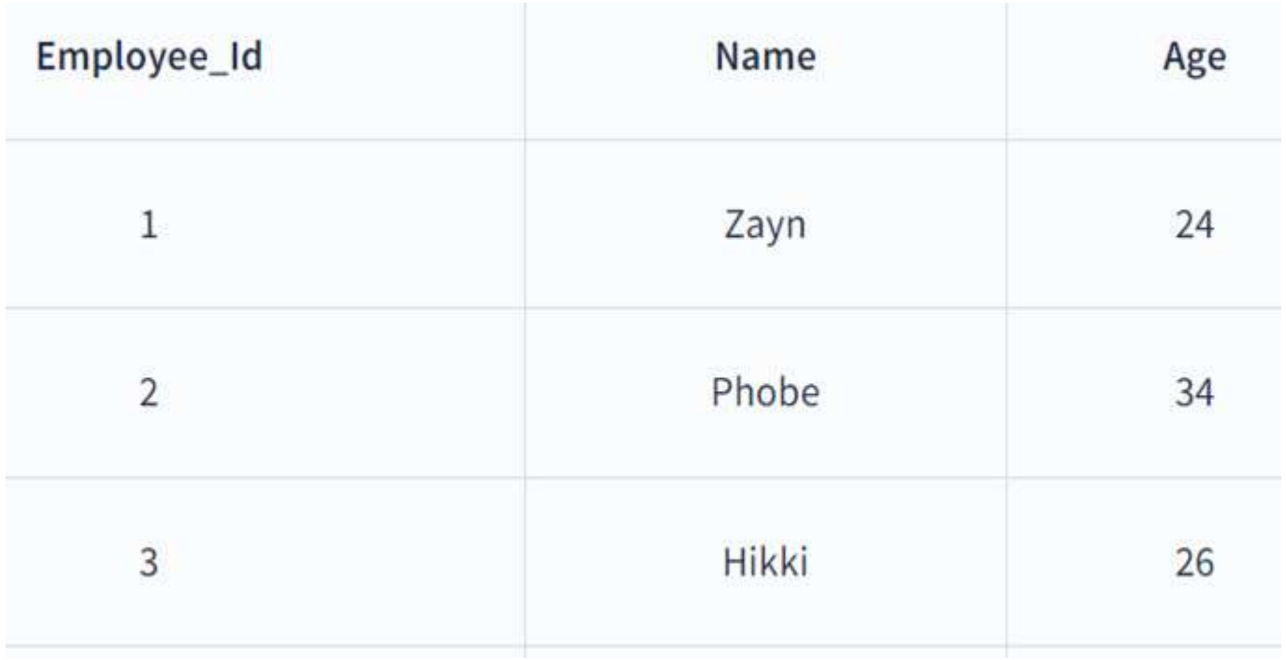
MULTI VALUED FUNCTIONAL DEPENDENCY

TRANSITIVE FUNCTIONAL DEPENDENCY

FULLY FUNCTIONAL DEPENDENCY

Trivial Functional Dependency:

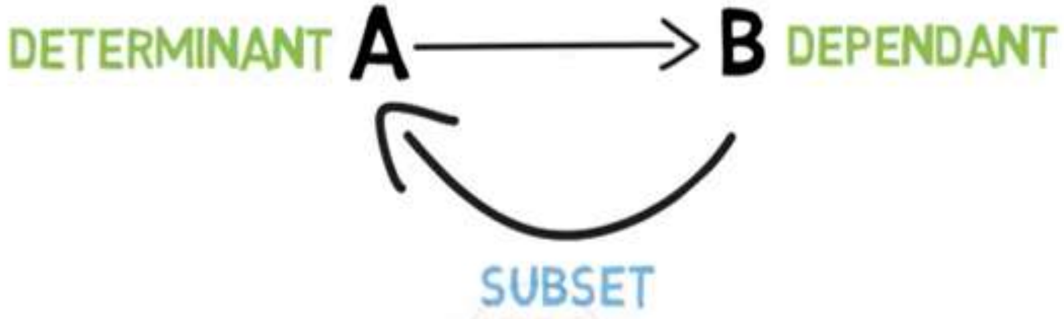
- *A functional dependency is called trivial if the attributes on the right side are the subset of the attributes on the left side of the functional dependency.*



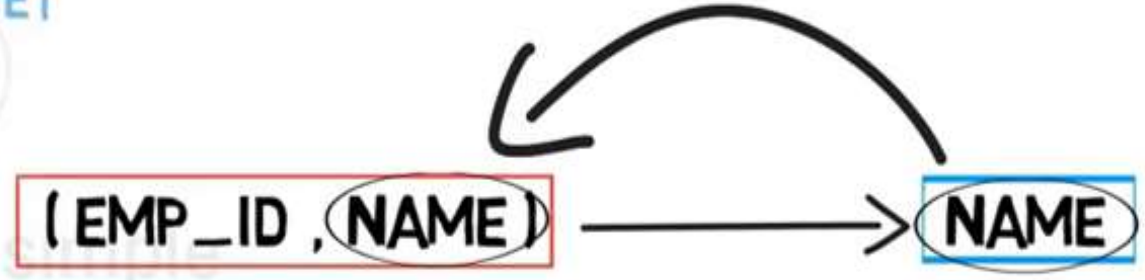
Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34
3	Hikki	26

{ Employee_Id, Name } → **{ Name }** is a Trivial functional dependency.
{ Employee_Id } → **{ Employee_Id }**, **{ Name }** → **{ Name }**, **{ Age }** → **{ Age }** are also Trivial.

TRIVIAL FUNCTIONAL DEPENDENCY



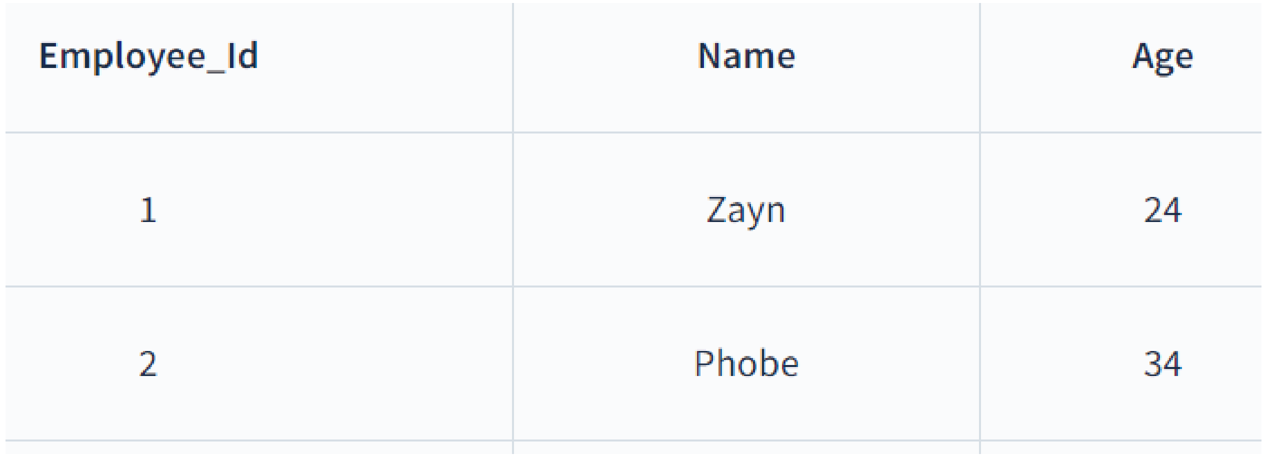
Emp_ID	Name	Age
1	XYZ	54
2	MNO	25
3	ABC	28
4	RST	25
5	ABC	35
6	GHI	22



TRIVIAL FUNCTIONAL DEPENDENCY

Non Trivial Functional Dependency

- It is the opposite of Trivial functional dependency. Formally speaking, in **Non-Trivial functional dependency**, dependent if **not a subset** of the determinant.
- **X** → **Y** is called a Non-trivial functional dependency if **Y** is **not a subset** of **X**. So, a functional dependency **X** → **Y** where **X** is a set of attributes and **Y** is also a set of the attribute but not a subset of **X**, then it is called ***Non-trivial functional dependency***.



Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34

- Here, **{ Employee_Id }** → **{ Name }** is a non-trivial functional dependency because **Name**(*dependent*) is **not a subset** of **Employee_Id**(*determinant*).
- Similarly, **{ Employee_Id, Name }** → **{ Age }** is also a non-trivial functional dependency.

NON TRIVIAL FUNCTIONAL DEPENDENCY

$$\begin{tikzcd}\text{DETERMINANT} & A \arrow[r] & B \arrow[l, bend left, "\text{NOT A SUBSET}"] & \text{DEPENDANT}\end{tikzcd}$$

EMP_ID $\longrightarrow$ NAME

EMP_ID $\longrightarrow$ AGE

VALID FUNCTIONAL DEPENDENCY

$$\begin{tikzcd}(\text{EMP\_ID} , \text{NAME}) \arrow[r] & \text{AGE} \arrow[l, bend left, "\text{NOT A SUBSET}"]\end{tikzcd}$$

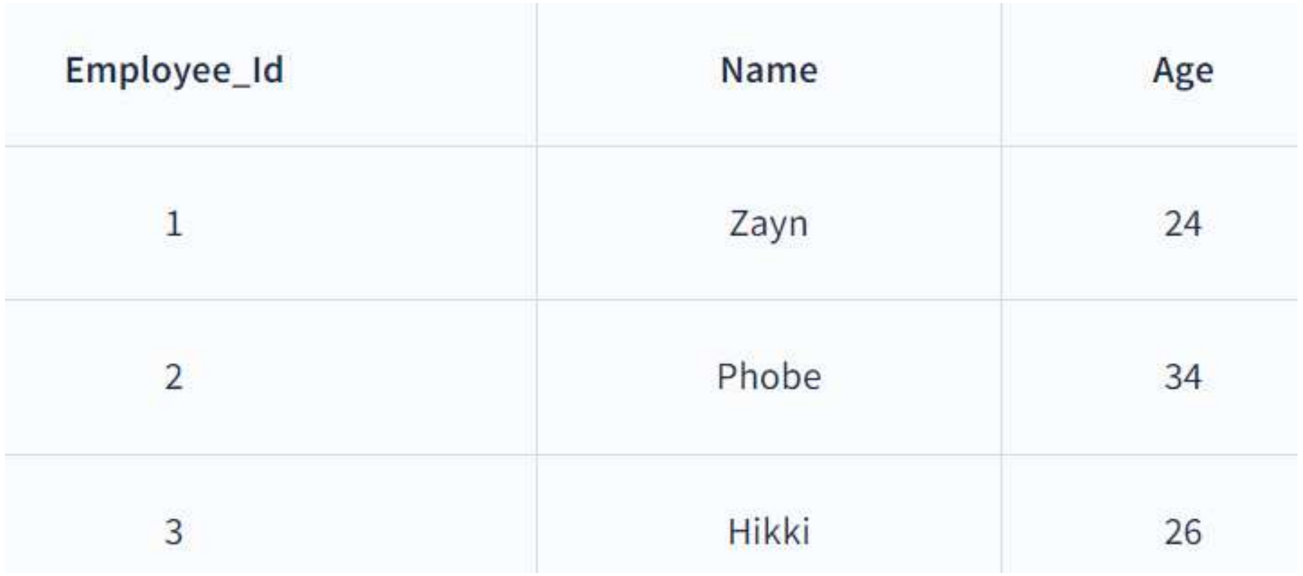
Emp_ID	Name	Age
1	XYZ	54
2	MNO	25
3	ABC	28
4	RST	25
5	ABC	35
6	GHI	22

Employee Table

NON TRIVIAL FUNCTIONAL DEPENDENCY

Multivalued Functional Dependency in DBMS

- In **Multivalued functional dependency**, attributes in the dependent set are **not dependent** on each other.
- For example, **X → { Y, Z }**, if there exists is **no functional dependency dependency** between **Y and Z**, then it is called as *Multivalued functional dependency*.



Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34
3	Hikki	26

Here, { **Employee_Id** } → { **Name, Age }** is a Multivalued functional dependency, since the dependent attributes **Name, Age** are not *functionally dependent*

MULTI VALUED FUNCTIONAL DEPENDENCY

$$\boxed{A \longrightarrow (B\ ,\ C)}$$

$$B \longrightarrow C$$

$$C \longrightarrow B$$

MULTI VALUED FUNCTIONAL DEPENDENCY

Emp_ID	Name	Age
1	XYZ	54
2	MNO	25
3	ABC	28
4	RST	25
5	ABC	35
6	GHI	22

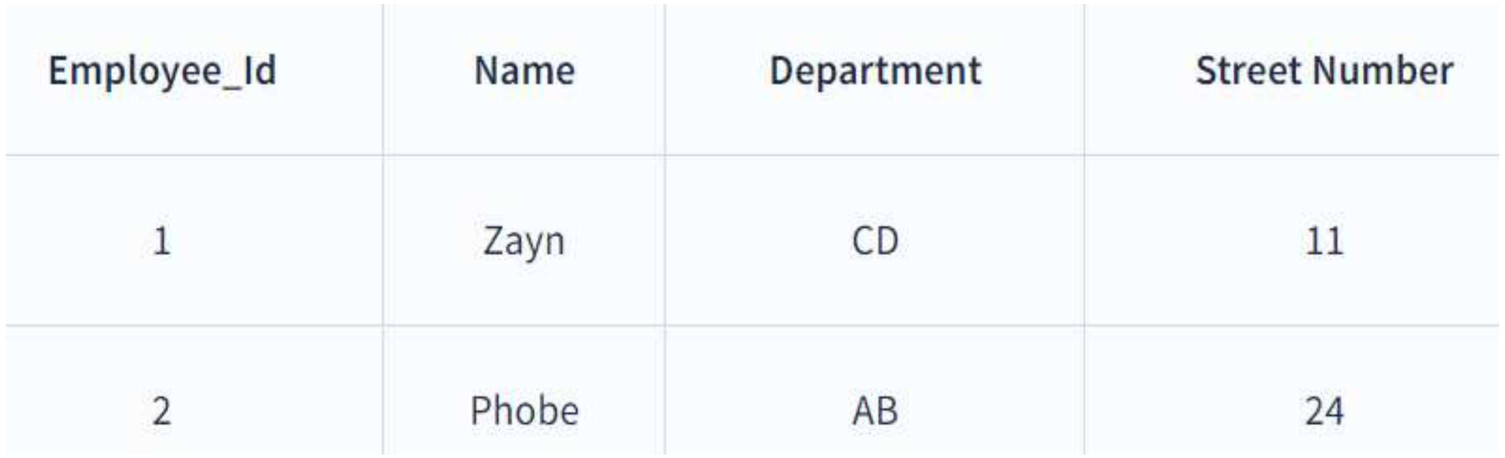
$$\boxed{EMP\_ID \longrightarrow (NAME\ ,AGE)}$$

$$NAME \nrightarrow AGE$$

$$AGE \nrightarrow NAME$$

Transitive Functional Dependency in DBMS

- Consider two functional dependencies **A → B** and **B → C** then according to the *transitivity axiom* **A → C** must also exist. This is called a transitive functional dependency.



Employee_Id	Name	Department	Street Number
1	Zayn	CD	11
2	Phobe	AB	24

Here, **{ Employee_Id → Department }** and **{ Department → Street Number}** holds true. *Hence*, according to the **axiom of transitivity**, **{ Employee_Id → Street Number }** is a valid functional dependency

TRANSITIVE FUNCTIONAL DEPENDENCY

$A \longrightarrow B$ VALID $\quad B \longrightarrow C$ VALID

$A \longrightarrow C$

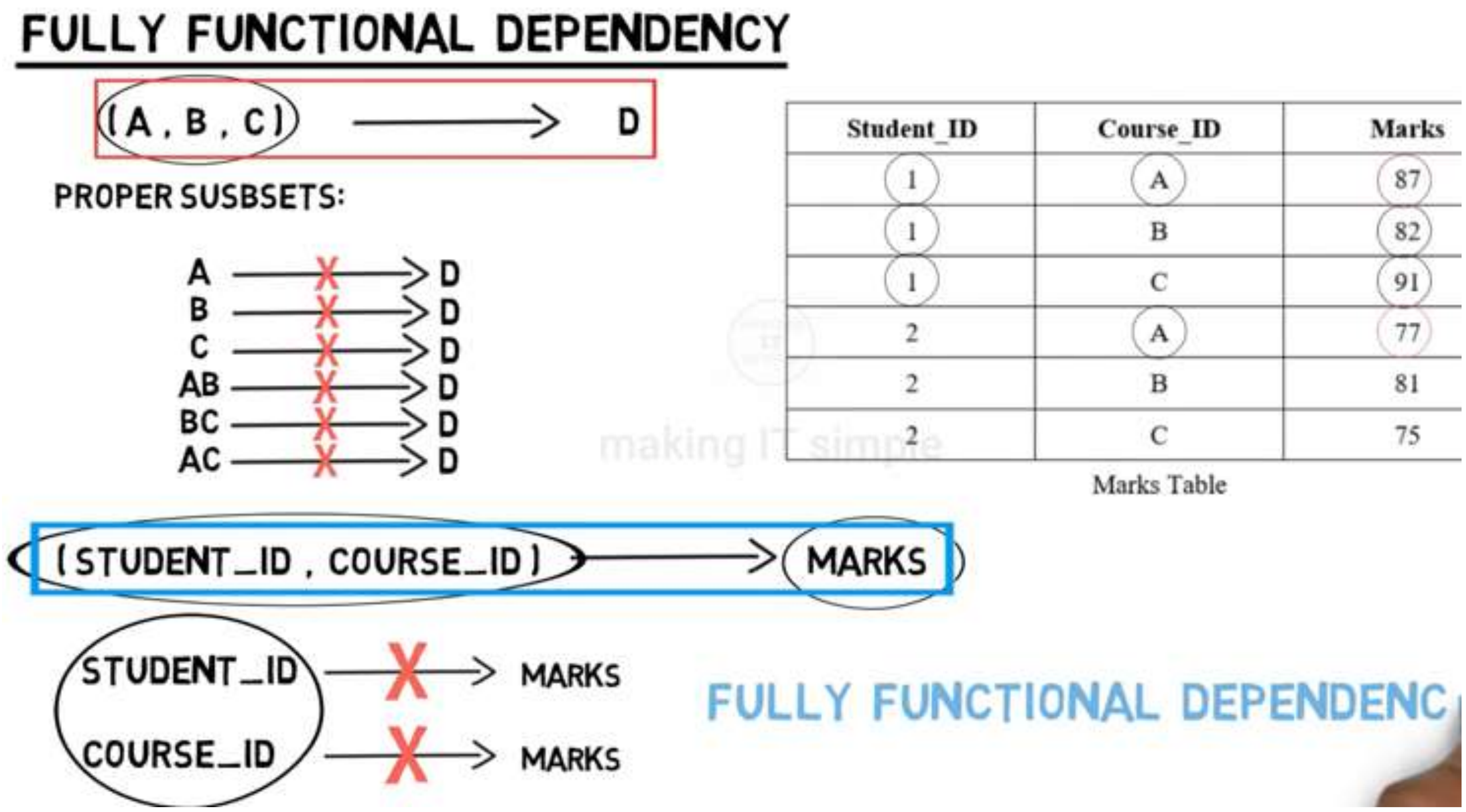
EMP_ID → DEPT_NAME DEPT_NAME → DEPT_BUILDING

EMP_ID → DEPT_BUILDING

Emp_ID	Name	Dept_Name	Dept_Buliding
1	XYZ	R & D	Building A
2	MNO	Design	Building B
3	ABC	Production	Building C

Fully Functional Dependency:

An attribute is fully functional dependent on another attribute, if it is FD on that attribute and not on any of its proper subset.



Student_ID	Course_ID	Marks
1	A	87
1	B	82
1	C	91
2	A	77
2	B	81
2	C	75

Marks Table

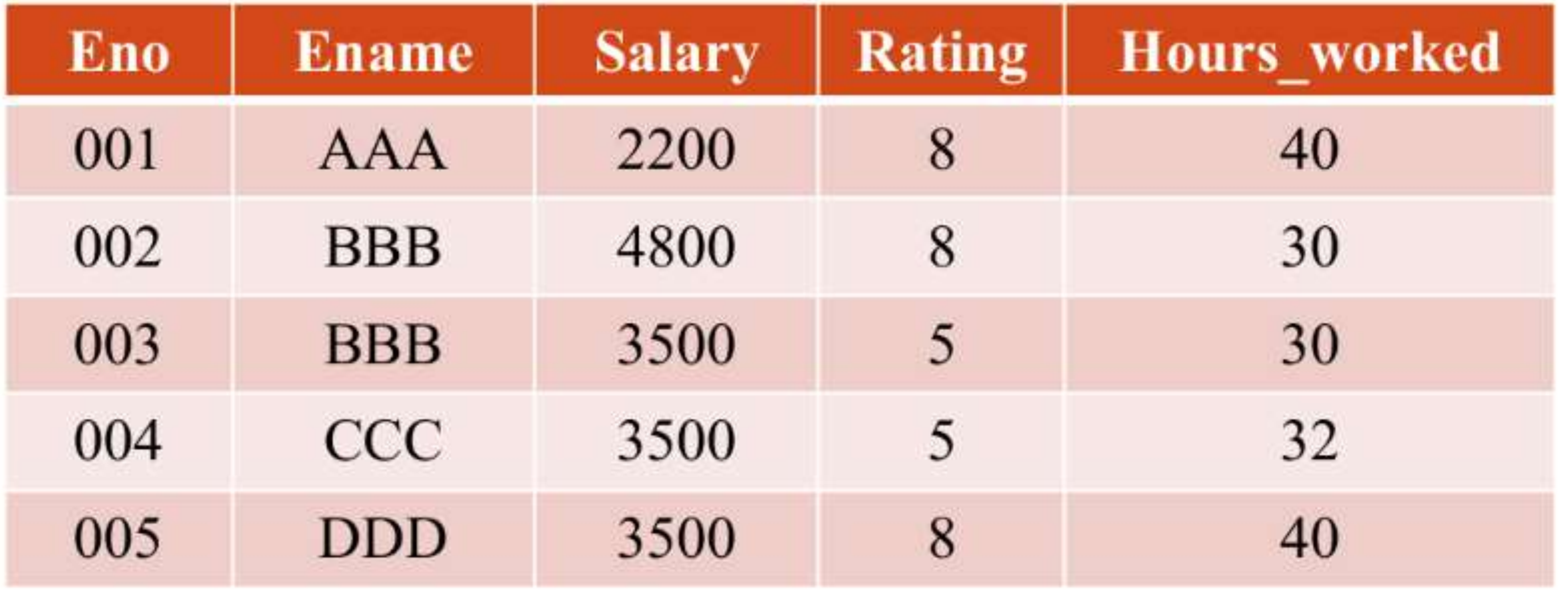
Decompositions

- Redundancy arises when relational schema forces an association between two or more attributes.
- Schema Refinement is the solution for many problems arising due to redundancy, and is solved by replacing the relation into a collection of smaller relations.
- Decomposition of a relation schema R replaces the relation schema by two or more relation schemas, that each consists the subsets of attributes of R and together includes all attributes of R.

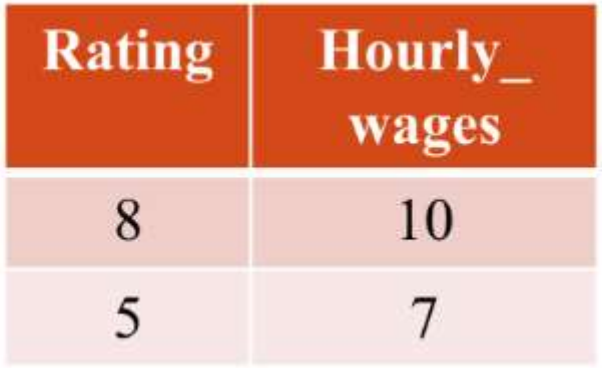
Example: Consider the hourly_emp table

Eno	Ename	Salary	Rating	Hourly_wages	Hours_worked
001	AAA	2200	8	10	40
002	BBB	4800	8	10	30
003	BBB	3500	5	7	30
004	CCC	3500	5	7	32
005	DDD	3500	8	10	40

- The above relation R is decomposed into two smaller relations namely hourly_empsd and wages.
- As a result, updating any one hourly_wages tuple associated with a rating in wages relation, involves updating single wage tuple. So this is more efficient than updating several tuples in hourly_emp relation and eliminates inconsistency.



Eno	Ename	Salary	Rating	Hours_worked
001	AAA	2200	8	40
002	BBB	4800	8	30
003	BBB	3500	5	30
004	CCC	3500	5	32
005	DDD	3500	8	40



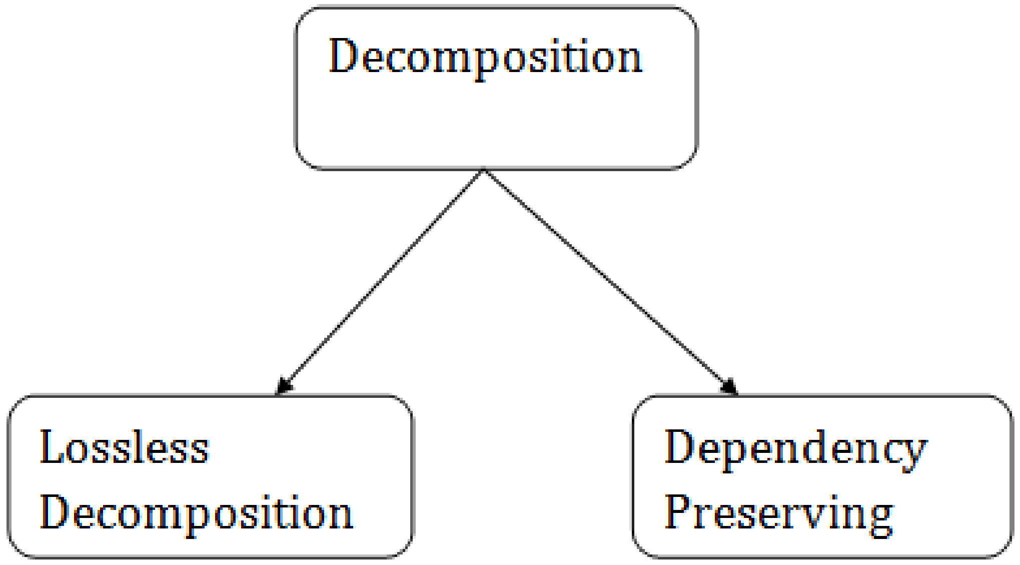
Rating	Hourly_ wages
8	10
5	7

Problems caused by Decomposition:

Decomposition may solve the problems of redundancy but may lead to some other problems.

- What are the problems caused by decomposition?
- When do we have to decompose a relation? -> [Normal Forms]

To answer the above first question, we have two properties



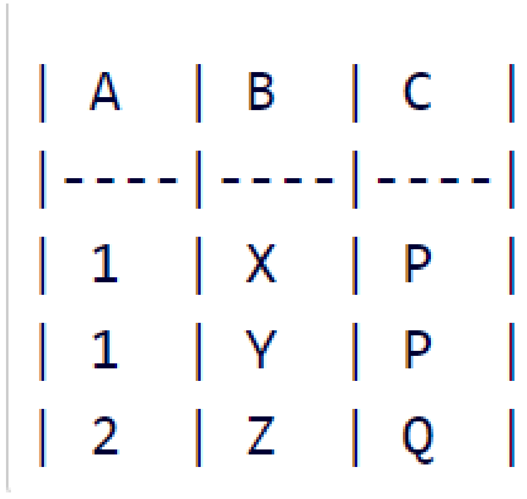
Lossless Decomposition

- When a relation is decomposed into two or more smaller relations, and the original relation can be perfectly reconstructed by taking the natural join of the decomposed relations, then it is termed as **lossless decomposition**. If not, it is termed "**lossy decomposition**.“

Example:

- Let's consider a table **\`R(A, B, C)\`** with a dependency **\`A → B\`**.
- If you decompose it into **\`R1(A, B)\`** and **\`R2(B, C)\`**, it would be lossy because you can't recreate the original table using natural joins.

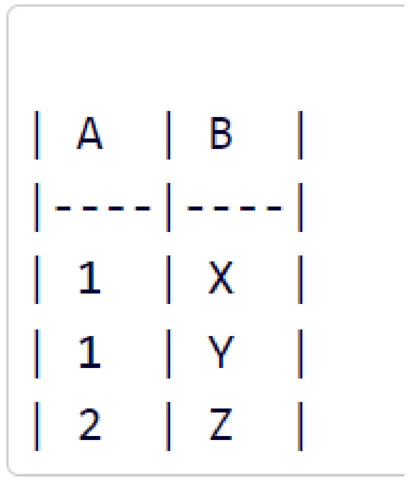
R(A,B,C)



A	B	C
1	X	P
1	Y	P
2	Z	Q

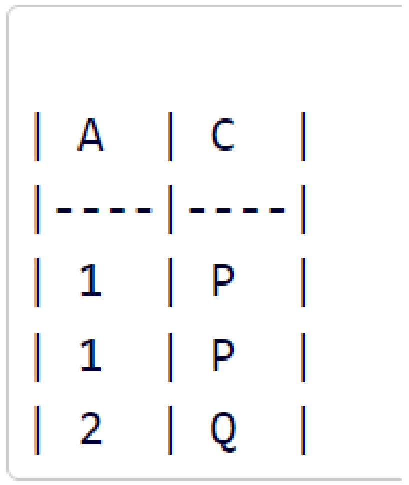
R into R1(A,B) and R2(A,C).

R1(A, B):



A	B
1	X
1	Y
2	Z

R2(A, C):



A	C
1	P
1	P
2	Q

Now, if we take the natural join of R1 and R2 on attribute A, we get back the original relation R. Therefore, this is a lossless decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD

  A->BC is a part of relation R1(ABC).

<EmployeeProjectDetail>

Emp_Code	Emp_Name	Emp_Email	Proj_Name	Proj_ID
101	John	j@demo.com	Project103	P03
101	John	jo@demo.com	Project101	P01
102	Ryan	r@exam.com	Project104	P04
103	Stephanie	st@abc.com	Project102	P02

In this relation we have the following FDs:

- Emp_Code -> {Emp_Name, Emp_Email}
- Proj_ID - > Proj_Name

Now, after decomposing the relation into EmployeeProject and ProjectDetail as:

<EmployeeProject> :

Emp_Code -> {Emp_Name, Emp_Email}

Emp_Code	Proj_ID	Emp_Name	Emp_Email
101	P03	John	j@demo.com
101	P01	John	jo@demo.com
102	P04	Ryan	r@exam.com
103	P02	Stephanie	st@abc.com

<ProjectDetail>:

Proj_ID - > Proj_Name

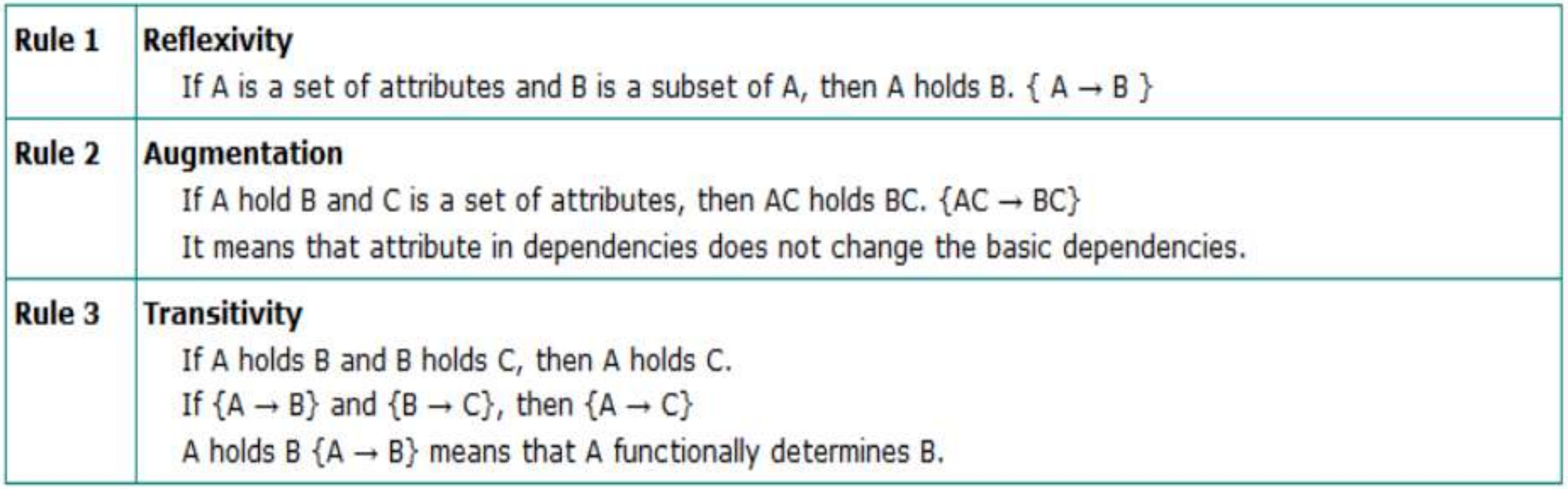
Proj_ID	Proj_Name
P03	Project103
P01	Project101
P04	Project104
P02	Project102

Armstrong Axioms/ Reasoning about FD’s:

- Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

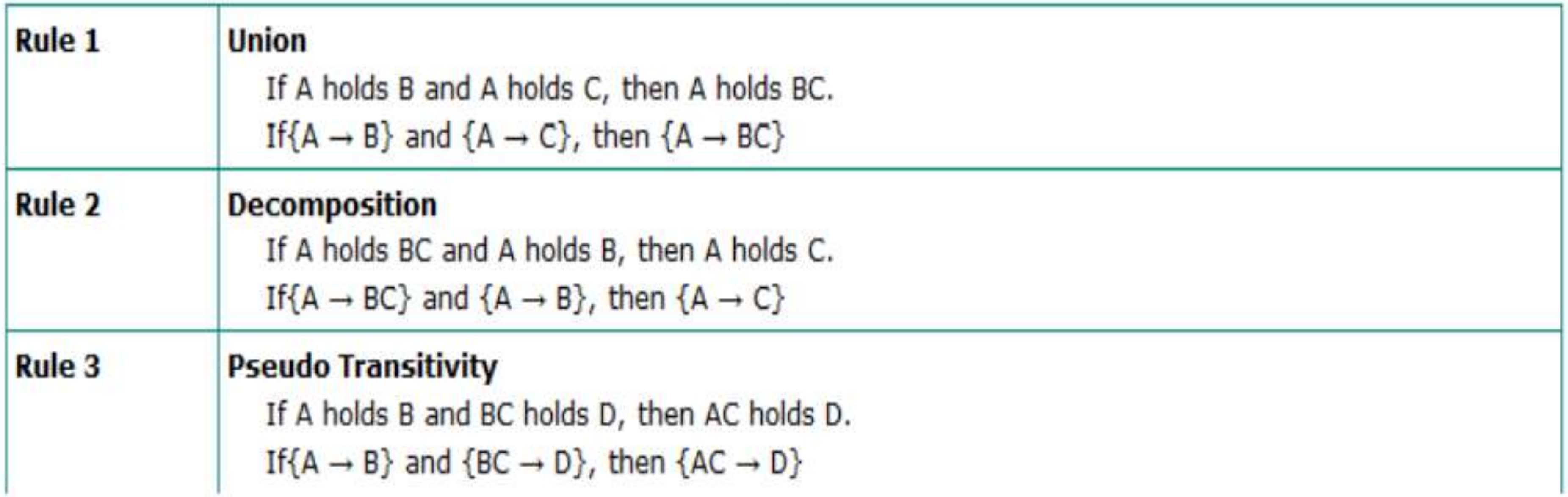
Various axioms rules or inference rules:

- **Primary Axioms:**



Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

- **Secondary or derived axioms:**

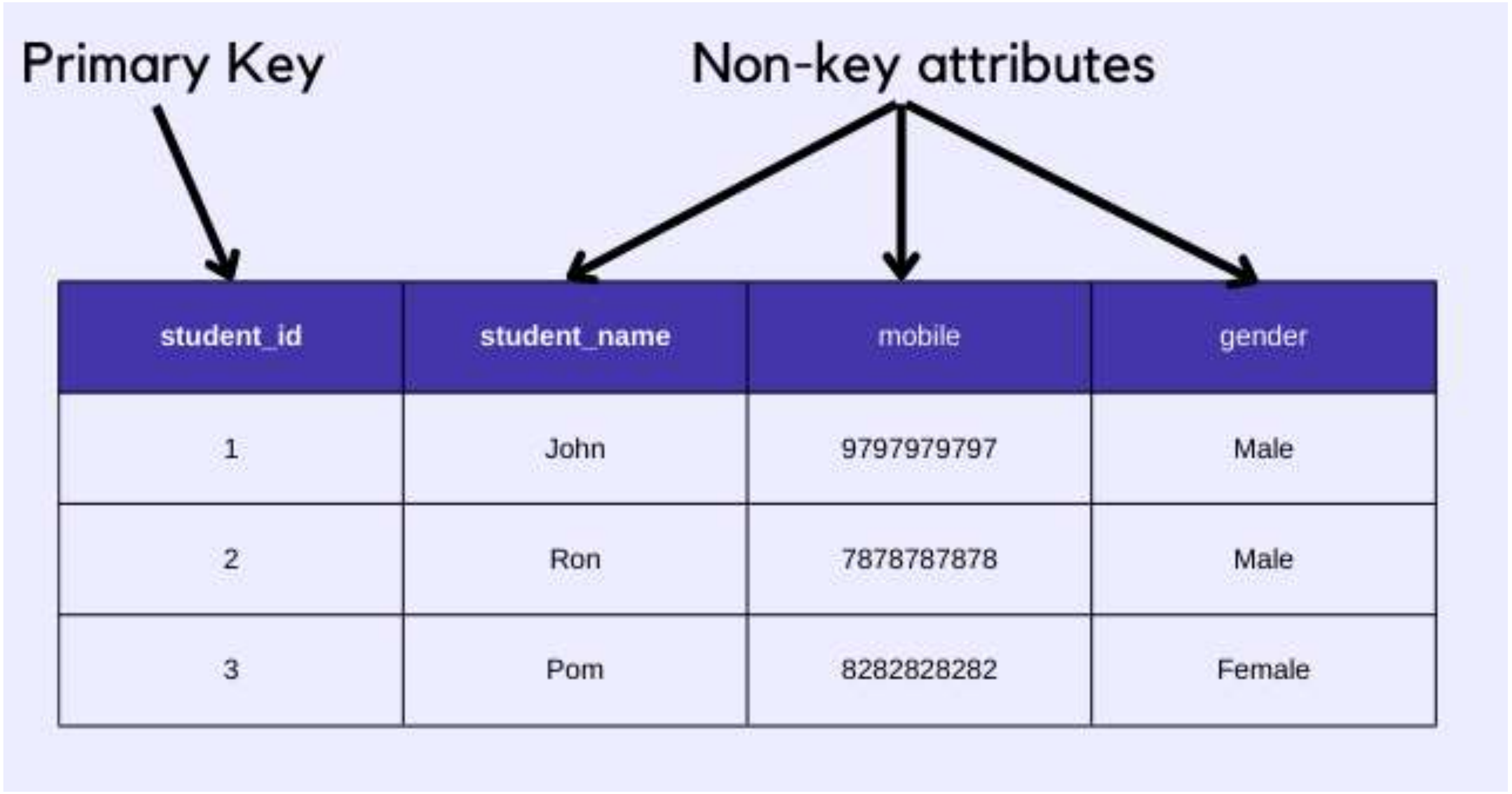


Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{$A \rightarrow B$} and {$A \rightarrow C$}, then {$A \rightarrow BC$}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{$A \rightarrow BC$} and {$A \rightarrow B$}, then {$A \rightarrow C$}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{$A \rightarrow B$} and {$BC \rightarrow D$}, then {$AC \rightarrow D$}

- **Attribute Closure (X^+):** Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it. It is also used to find out the Candidate Keys.

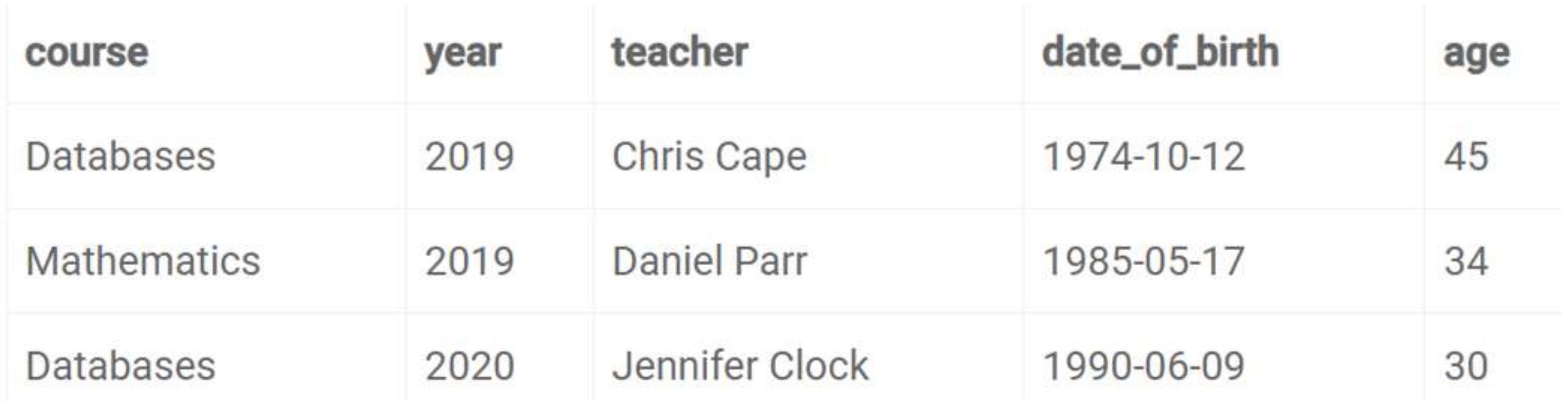
- **Prime and non-prime attributes:** Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.
- **Candidate Key**: Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely. Consider student table: student(sno, sname, sphone,age) we can take sno as candidate key. We can have more than 1 candidate key in a table.
  - Types of candidate keys:
    1. simple(having only one attribute)
    2. composite(having multiple attributes as candidate key)
- **Super Key**: Super Key is set of attributes of a relation which can be used to identify a tuple uniquely. Consider student table: student(sno, sname,sphone,age) we can take sno, (sno, sname) as super key

Primary Key and Non-key attributes



student_id	student_name	mobile	gender
1	John	9797979797	Male
2	Ron	7878787878	Male
3	Pom	8282828282	Female

Finding the Attribute Closure



course	year	teacher	date_of_birth	age
Databases	2019	Chris Cape	1974-10-12	45
Mathematics	2019	Daniel Parr	1985-05-17	34
Databases	2020	Jennifer Clock	1990-06-09	30

Here are the functional dependencies in this table:

- *course*, *year* -> *teacher*
  - Given the course and year, you can determine the teacher who taught the course that year.
- *teacher* -> *date_of_birth*
  - Given a teacher, you can determine the teacher’s date of birth.
- *year*, *date_of_birth* -> *age*
  - Given the year and date of birth, you can determine the age of the teacher at the time the course was taught.

- First, consider the closure of a set **{year}, denoted {year}**$^+$
- The first functional dependency ***[course, year -> teacher]*** requires the course in addition to the year, so it doesn't add anything to **{year}**$^+$**.**
- The functional dependency ***[year, date_of_birth -> age]*** requires the date of birth in addition to the year, so it doesn't add anything to **{year}**$^+$ either. So, **{year}**$^+$ contains only one column, year, that is **{year}**$^+$ **= {year}.**
- Next, let's look at **{year, teacher}**$^+$**.** If I know the teacher, I also know the date of birth because of the ***teacher ->date_of_birth*** functional dependency.
- So, **date_of_birth** is also in **{year, teacher}**$^+$**,** and I know three columns**:** **{*year, teacher, date_of_birth*}.**
- If I know the year and date of birth, I can also determine the age. Now, **{year, teacher}**$^+$ has four columns **{*year, teacher, date_of_birth, age*}.**

Example1:

We are given the relation R(A, B, C, D, E). This means that the table R has five columns: A, B, C, D, and E. We are also given the set of functional dependencies: {A->B, B->C, C->D, D->E}.

What is $\{A\}^+$?

Sol:

$\{A\}^+ \rightarrow \{A\}$	$[A \rightarrow A]$
$\rightarrow \{A,B\}$	$[A \rightarrow B]$
$\rightarrow \{A,B,C\}$	$[B \rightarrow C]$
$\rightarrow \{A,B,C,D\}$	$[C \rightarrow D]$
$\{A\}^+ \rightarrow \{A,B,C,D,E\}$	$[D \rightarrow E]$

Example 2

- Let's look at another example. We are given R(A, B, C, D, E, F). The functional dependencies are {AB->C, BC->AD, D->E, CF->B}.

What is $\{A, B\}^+$?

Sol:

$\{A,B\}^+$ □ {A,B}

□ {A,B,C}

□ {A,B,C,D}

$\{A,B\}^+$□ {A,B,C,D,E}

Question: The following functional dependencies are given:

{AB→CD, AF→D, DE→F, C→G, F→E, G→A}

Which one of the following options is false? (**GATE 2006)**

A. CF+ = {ACDEFG}
B. BG+ = {ABCDG}
C. AF+ = {ACDEFG}
D. AB+ = {ABCDFG}

Solution:

If we take the attribute closure of option A, we will get, $(CF)^+$= {ACDEFG}

If we take the attribute closure of option B, we will get,$(BD)^+$={ABCDG}

This can be done with the steps discussed above in the article.

But option C and D have attribute closure: $(AF)^+$=(AFDE) and $(AB)^+$=(ABCDG).

Therefore, options C and D are false.

Example 2: Find candidate keys for R(ABCDE) having following FD's

$A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A$

Solution:

$A^+ = \{ABCDE\}$ A is candidate key and prime attribute

$E \rightarrow A$ so replace A by E

$E^+ = \{ABCDE\}$ E is candidate key and prime attribute

$CD \rightarrow E$ replace E by CD

$CD^+ = \{ABCDE\}$ ($C^+ = C$ and $D^+ = D$) no proper subset of CD is superkey. so CD is candidate key

$B \rightarrow D$

$BC^+ = \{ABCDE\}$ ($B^+ = BD$) BC is candidate key

A, E, CD, BC are candidate keys

GATE Question: Consider a relation scheme **R = (A, B, C, D, E, H)** on which the following functional dependencies hold: **{A–>B, BC–>D, E–>C, D–>A}**. What are the candidate keys of R?

1. AE, BE
2. AE, BE, DE
3. AEH, BEH, BCH
4. AEH, BEH, DEH

Solution: A set of attributes S is a candidate key of relation R if the closure of S is all attributes of R and there is no subset of S whose closure is all attributes of R.

If we look closely, attributes E and H are not determined by any of the dependencies given in the FD set. Therefore they need to be present on the left-hand side. Only option D satisfies the given condition. Therefore, we can check for attribute closure.

$(AEH)^{+}=\{ABCDEH\}$

$(BEH)^{+}=\{ABCDEH\}$

$(DEH)^{+}=\{ABCDEH\}$

Option D is correct.

Question 1 : Given a relation R(ABCDEF) having FDs {AB→C, C→D, D→E , F→B, E→F} Identify the prime attributes and non prime attributes .

Solution :

$(AB)^+$: {ABCDEF} ⇒ Super Key
$(A)^+$: {A} ⇒ Not Super Key
$(B)^+$: {B} ⇒ Not Super Key

Prime Attributes : {A,B}

(AB) → Candidate Key
↓ (as F → B)

$(AF)^+$: {AFBCDE}
$(A)^+$: {A} ⇒ Not Super key
$(F)^+$: {FB} ⇒ Not Super Key

(AF) → Candidate Key
↓

$(AE)^+$: {AEFBCD}
$(A)^+$: {A} ⇒ Not Super key
$(E)^+$: {EFB} ⇒ Not Super key

(AE) → Candidate Key
↓

$(AD)^+$: {ADEFBC}
$(A)^+$: {A} ⇒ Not Super key
$(D)^+$: {DEFB} ⇒ Not Super key

(AD) → Candidate Key
↓

$(AC)^+$: {ACDEFB}
$(A)^+$: {A} ⇒ Not Super Key
$(C)^+$: {DCEFB} ⇒ Not Super Key

⇒ Candidate Keys {AB, AF, AE, AD, AC}
⇒ Prime Attributes {A,B,C,D,E,F}
⇒ Non Prime Attributes {}

Question 2: Given a relation R(ABCDEF) having FDs {AB → C, C → DE , E → F, C → B} Identify the prime attributes and non prime attrib utes.

Solution :

$(AB)^+$: {A B C D E F}

$(A)^+$: {A}

$(B)^+$: {B}

(AB) ⇒ (AC), $(AC)^+$: {ABCDEF}

$(C)^+$: {DECBF}

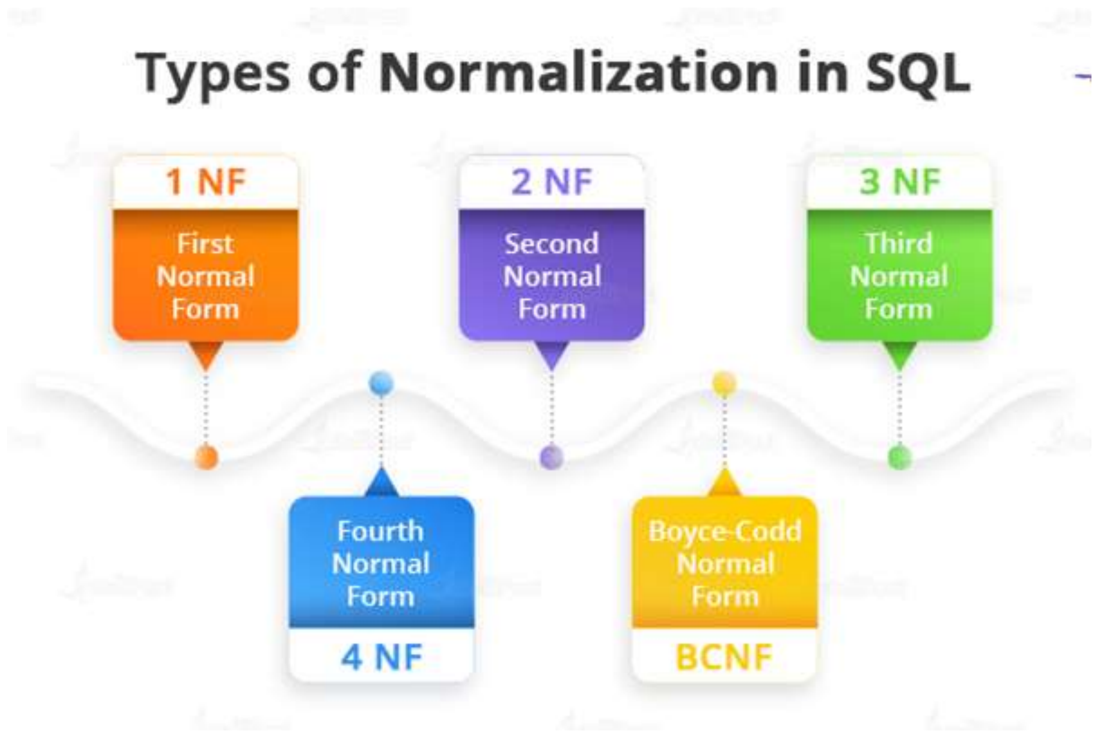
⇒ Candidate Keys {AB, AC}

⇒ Prime Attributes {A,B,C}

⇒ Non Prime Attributes {D,E,F}

Normalization

- Normalization in DBMS is a technique using which you can organize the data in the database tables so that:
  - There is less repetition of data,
  - A large set of data is structured into a bunch of smaller tables,
  - and the tables have a proper relationship between them.
- Types of Normalization:



1st Normal Form (1NF):

- There are 4 basic rules that a table should follow to be in 1st Normal Form

Rule 1:

□Each column should contain atomic value

□Entries like X,Y and W,X violates the rule

Rule 2:

□A column should contain values that are of the same type

□Do not inter-mix different types of values in any column

Rule 3:

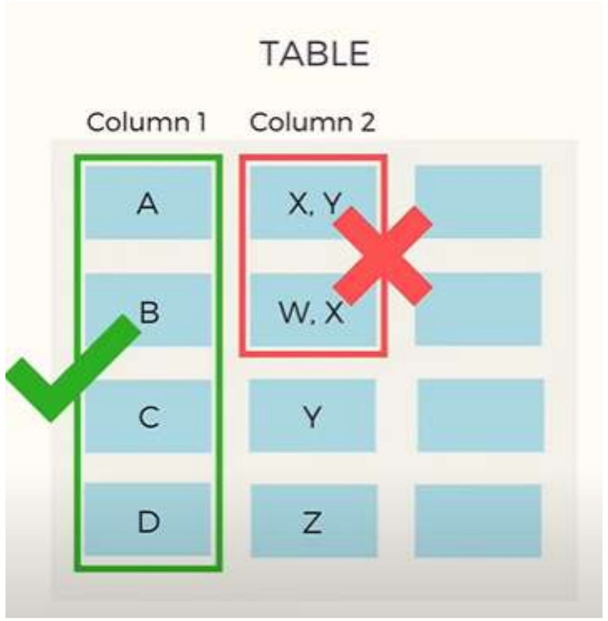
□Each column should have a Unique name

□Same names leads to confusion at the time of data retrieval.

Rule 4:

□Order of which data is saved doesn’t matter

□Using SQL query, you can easily fetch the data in any order from a table

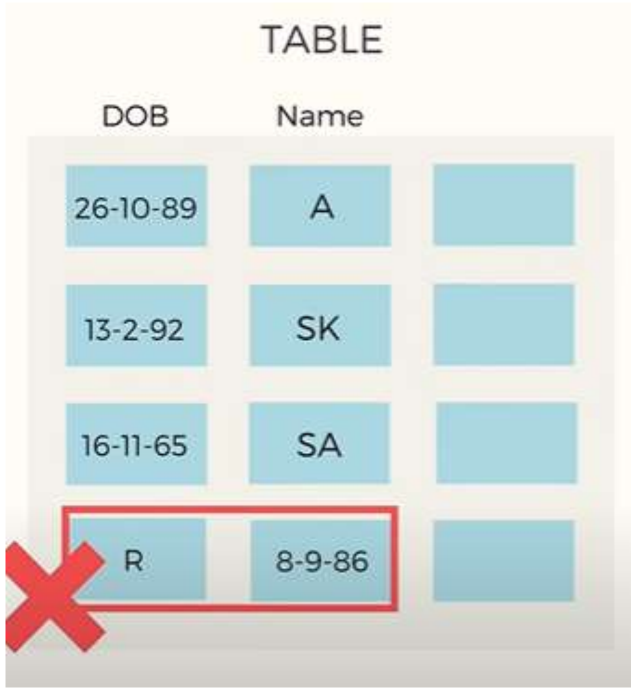


TABLE

Column 1	Column 2	
A	X, Y	
B	W, X	
C	Y	
D	Z	

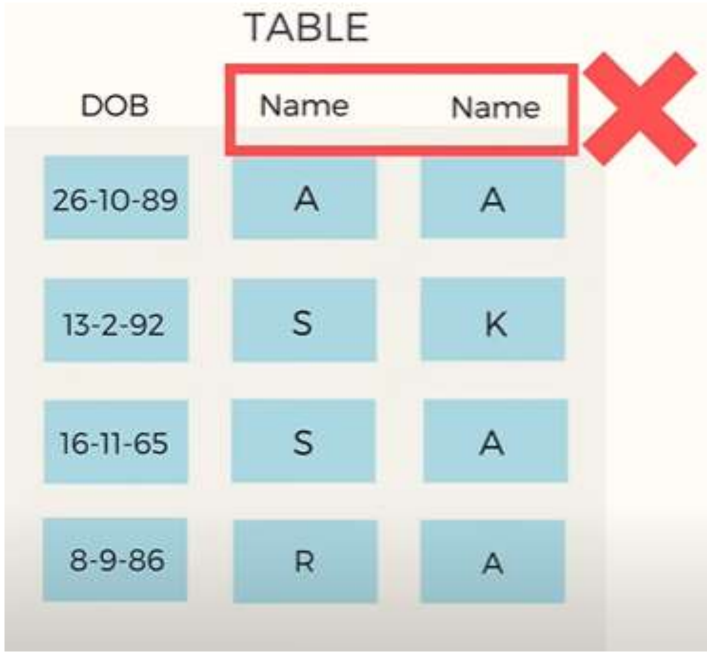
RULE 1

RULE 2



TABLE

DOB	Name	
26-10-89	A	
13-2-92	SK	
16-11-65	SA	
R	8-9-86	

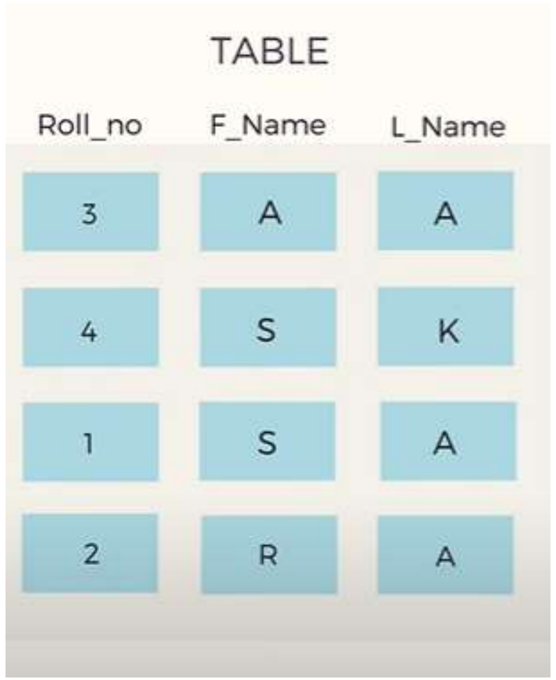


TABLE

DOB	Name	Name
26-10-89	A	A
13-2-92	S	K
16-11-65	S	A
8-9-86	R	A

RULE 3

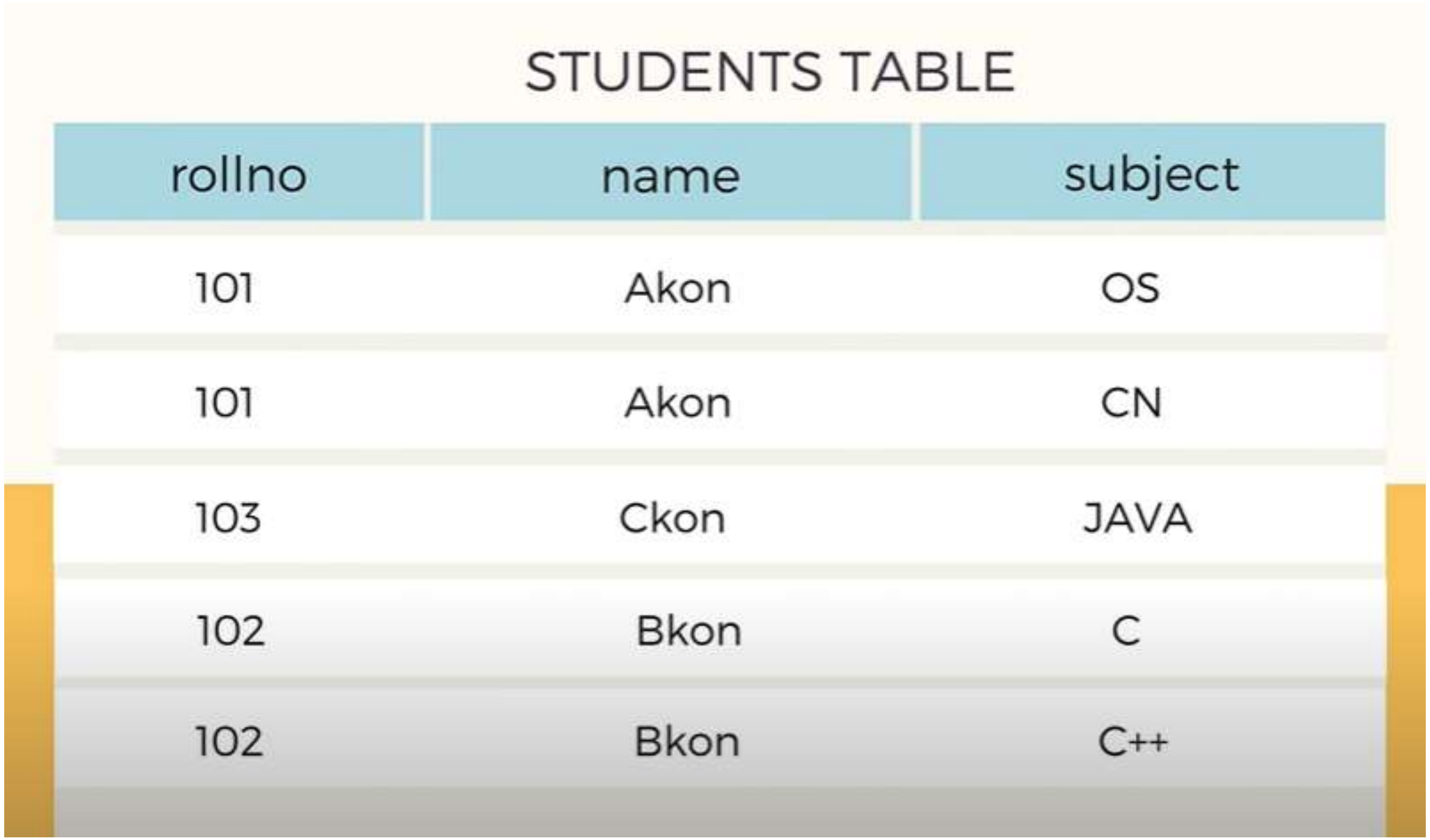
RULE 4



TABLE

Roll_no	F_Name	L_Name
3	A	A
4	S	K
1	S	A
2	R	A

EXAMPLE:



STUDENTS TABLE

rollno	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	JAVA
102	Bkon	C
102	Bkon	C++

EXAMPLE:

1st Normal Form

emp_id	emp_name	emp_mob	emp_skills
1	John Tick	9999957773	Python, JavaScript
2	Darth Trader	8888853337	HTML, CSS, JavaScript
3	Rony Shark	7777720008	Java, Linux, C++

emp_id	emp_name	emp_mob	emp_skill
1	John Tick	9999957773	Python
1	John Tick	9999957773	JavaScript
2	Darth Trader	8888853337	HTML
2	Darth Trader	8888853337	CSS
2	Darth Trader	8888853337	JavaScript
3	Rony Shark	7777720008	Java
3	Rony Shark	7777720008	Linux
3	Rony Shark	7777720008	C++

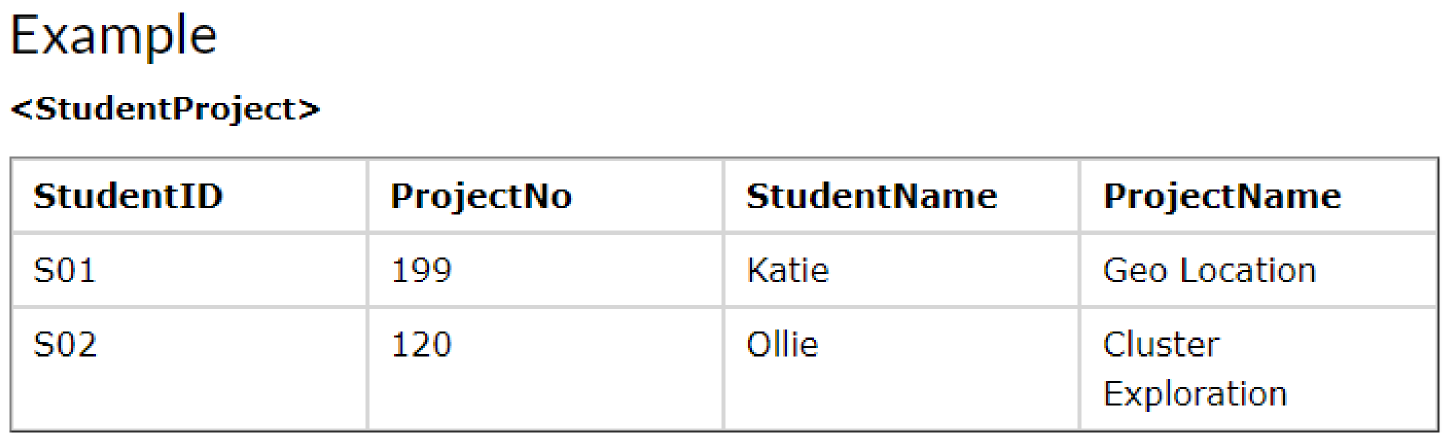
2nd Normal Form

There are 2 basic rules that a table should follow to be in 2nd Normal Form

- It should be in 1st normal form
- the table should not have “Partial Dependency”

What is Partial Dependency?

Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key.



Example

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

The prime key attributes are **StudentID** and **ProjectNo**, and

StudentID = Unique ID of the student**StudentName** = Name of the student**ProjectNo** = Unique ID of the project**ProjectName** = Name of the project

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a **candidate key**, to be Partial Dependent.

The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectNo**, which makes the relation Partial Dependent.

Therefore, the <StudentProject> relation violates the 2NF in Normalization and is considered a bad database design.

To remove Partial Dependency and violation on 2NF, decompose the tables –

<StudentInfo>

StudentID	ProjectNo	StudentName
S01	199	Katie
S02	120	Ollie

<ProjectInfo>

ProjectNo	ProjectName
199	Geo Location
120	Cluster Exploration

3rd Normal Form

A table is said to be in the Third Normal Form when,

1. It satisfies the First Normal Form and the Second Normal form.
2. And, it doesn't have Transitive Dependency.