Unit 4 Database Normalization

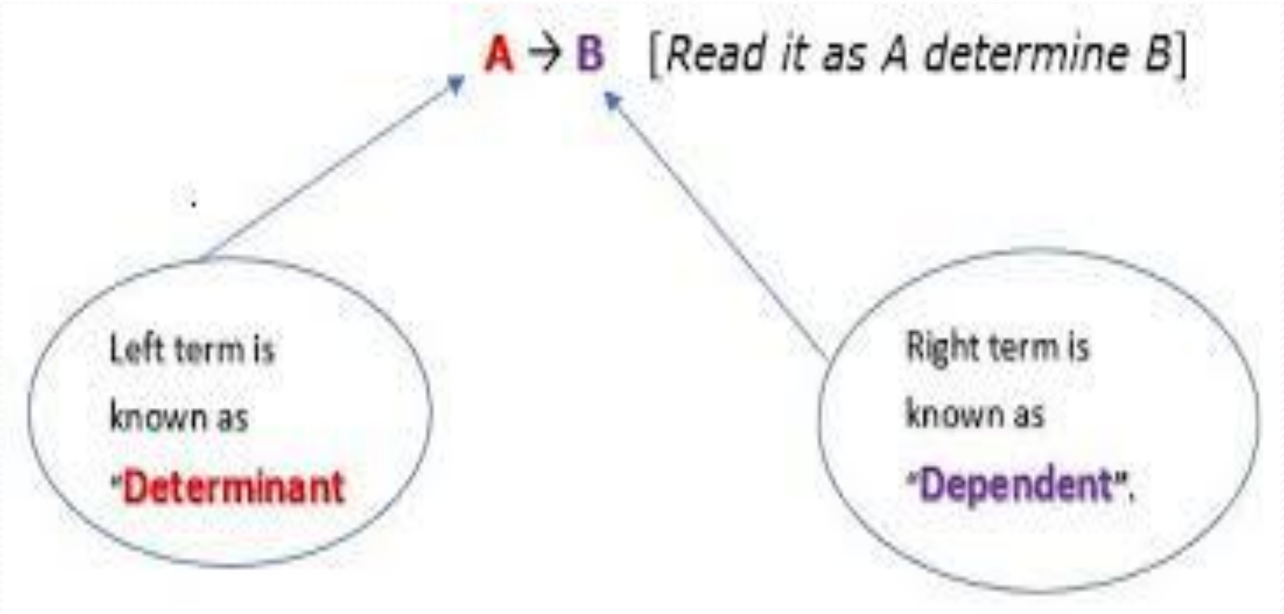
Definition and Importance of Normalization, Functional Dependencies. Normalization: 1NF, 2NF, 3NF, BNF.and 4NF.

Functional Dependency (FD)

- Functional Dependency determines the relation of one attribute to another attribute in a database management system (DBMS) system.
- Functional dependency helps you to maintain the quality of data in the database.
- A functional dependency is denoted by an arrow →.
- The functional dependency of X on Y is represented by

  $X \rightarrow Y$.
- Functional Dependency plays a vital role to find the difference between good and bad database design.
- The functional dependency is a relationship that exists between two attributes. 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 [Read it as A determine B]
Left term is known as "Determinant
Right term is known as "Dependent".

For example:

Assume we have an **Student** table with attributes: Roll_no, Name, GPA. Here Roll_no attribute can uniquely identify the Name attribute of **Student** table because if we know the Roll_no, we can tell that student name associated with it.

Functional dependency can be written as:

Roll_no → Name

We can say that Name is functionally dependent on Roll_no.

Roll_no	Name	GPA
1	Prabin	3
2	Suman	3
3	Suman	3.5
4	Tilak	2.5

A → B

Roll_no is determinant

Roll_no → Name

Roll_no →GPA

1 → Prabin

2 → Suman

3 → Suman

GPA is not determinant

GPA → NAME

3 → Prabin

3 → Suman

Types of Functional dependency

- **Trivial Functional Dependency**
- **Non-Trivial Functional Dependency**
- **Multivalued Dependency**
- **Transitive Dependency**

Trivial Functional dependency:

- In Trivial Functional Dependency, a dependent is always a subset of the determinant.
- $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
- i.e. If $A \rightarrow B$ and B is the subset of A, then it is called trivial functional dependency
- $A \cap B \neq \phi$
- Consider a table with two columns Employee_Id and E mployee_Name.
- {Employee_id, Employee_Name} → Employee_Na me is a trivial functional dependency, the dependent Employee_name is a subset of {Employee_Id, Employee_Name}

Non-trivial functional dependency

- In Non-trivial functional dependency, the dependent is strictly not a subset of the determinant.
- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
- i.e. If $A \rightarrow B$ and B is not a subset of A, then it is called Non-trivial functional dependency.
- When $A \cap B = \phi$, then $A \rightarrow B$ is called as complete non-trivial.
- Consider a table with three columns Employee_Id , Employee_Na me and age
- Here, Employee_Id → Employee_Name is a non-trivial functional dependency, since the dependent Employee_Name is not a subset of determinant Employee_Id
- Similarly, {Employee_Id , Employee_Name } → age is also a non-trivial functional dependency, since age is not a subset of {Employee_Id , Employee_Name}

Multivalued Functional Dependency

- In Multivalued functional dependency, entities of the dependent set are not dependent on each other. i.e. If a → {b, c} and there exists no functional dependency between b and c, then it is called a multivalued functional dependency.

roll_no	name	age
1	Ram	17
2	Hari	18
3	Ram	18

- Here, roll_no → {name, age} is a multivalued functional dependency, since the dependents name & age are not dependent on each other(i.e. name → age or age → name doesn't exist !)

Transitive Functional Dependency

- In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If a → b & b → c, then according to axiom of transitivity, a → c. This is a transitive functional dependency

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

- Here, enrol_no → dept and dept → building_no, Hence, according to the axiom of transitivity, enrol_no → building_no is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Inference Rule (IR)

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.
- The Functional dependency has 6 types of inference rule.

- **Reflexive Rule (IR_1)**
  - In the reflexive rule, if Y is a subset of X, then X determines Y.
  - If $X \supseteq Y$ then $X \rightarrow Y$
  - E.g SID → SID
- **Augmentation Rule (IR_2)**
  - The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.
  - If $X \rightarrow Y$ then $XZ \rightarrow YZ$
  - E.g SID → NAME then SID PHONE → NAME,PHONE
- **Transitive Rule (IR_3)**
  - In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.
  - If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
  - E.g SID → NAME and NAME → CITY then SID → CITY

- **Union Rule (IR_4)**
  - Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.
  - If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

- **Decomposition Rule (IR_5)**
  - Decomposition rule is also known as project rule. It is the reverse of union rule.
  - This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.
  - If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

- **Pseudo transitive Rule (IR_6)**
  - In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.
  - If $X \rightarrow Y$ and $Z \rightarrow W$ then $XZ \rightarrow YW$

Closure Of Functional Dependency

- The **Closure Of Functional Dependency** means the complete set of all possible attributes that can be functionally derived from given functional dependency using the inference rules known as Armstrong’s Rules.
- If “F” is a functional dependency then closure of functional dependency can be denoted using “$\{F\}^{+}$”.

Closure Of Functional Dependency…

- There are three steps to calculate closure of functional dependency. These are:
  - Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.
  - Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.
  - Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.

- **Example**

  X^+ → contains set of attributes determined by X

  X → set of attributes

  R(A,B,C,D,E}

  FD: {A →B,B →C,C →D,D →E}

  A^+={A,B,C,D,E}

  AD^+={A,D,B,C,E}

  B^+={B,C,D,E}

  CD^+={C,D,E}

  C+={C,D,E}

- **Super key**: set of attributes whose closure contains all the attributes of given relations. In the above relation **A** and **AD** are super key
- **Candidate key**: Minimal super key. Or Super key whose proper subset is not a super key. In the above relation **A** is candidate key

Dependency Preserving Decomposition

- A Decomposition D = { R_1, R_2, R_3….Rn } of R is dependency preserving with respect to a set F of Functional dependency if $(F_1 \cup F_2 \cup \dots \cup Fm)^+ = F^+$.

Consider a relation R

R → F{...with some functional dependency(FD)....}

R is decomposed or divided into R_1 with FD { f_1 } and R_2 with { f_2 }, then there can be three cases:

- $f_1 \cup f_2 = F$ → Decomposition is dependency preserving.
- $f_1 \cup f_2$ is a subset of F → Not Dependency preserving.
- $f_1 \cup f_2$ is a super set of F → This case is not possible.

Let R{A,B,C,D} with functional dependency

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

R is decomposed into R1{A,B},R2{B,C},R3{B,D}

R_1{A,B}

$A \to B$

R_2{B,C}

$B \to C$

$C \to B$

R_3{B,D}

$B \to D$

$D \to B$

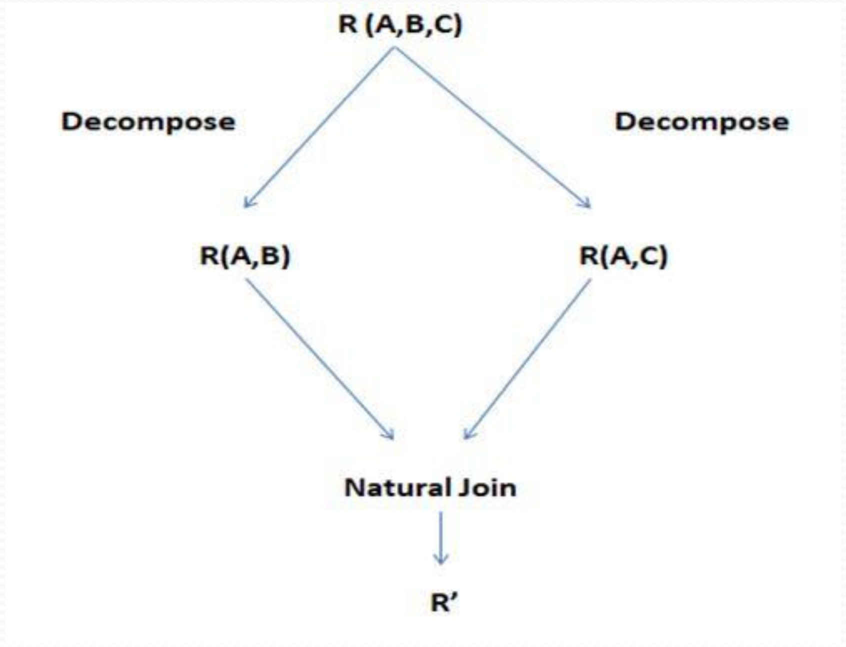
R_1{A,B}	R_2{B,C}	R_3{B,D}
~~$A \to A$~~	~~$B \to B$~~	~~$B \to B$~~
~~$B \to B$~~	~~$C \to C$~~	~~$D \to D$~~
$A \to B$	$B \to C$	$B \to D$
~~$B \to A$~~	$C \to B$	$D \to B$

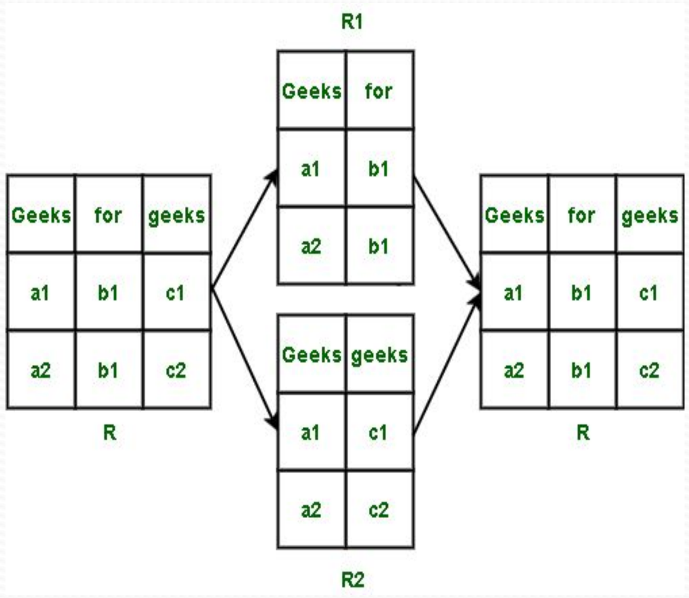
Now

$A \to B$, $B \to C$, $C \to B$, $B \to D$, $D \to B$ [C+=CBD]

Lossless and Lossy Decomposition in DBMS

- Decomposition is a process of dividing a single relation into two or more sub relations.





R

Geeks	for	geeks
a1	b1	c1
a2	b1	c2

R1

Geeks	for
a1	b1
a2	b1

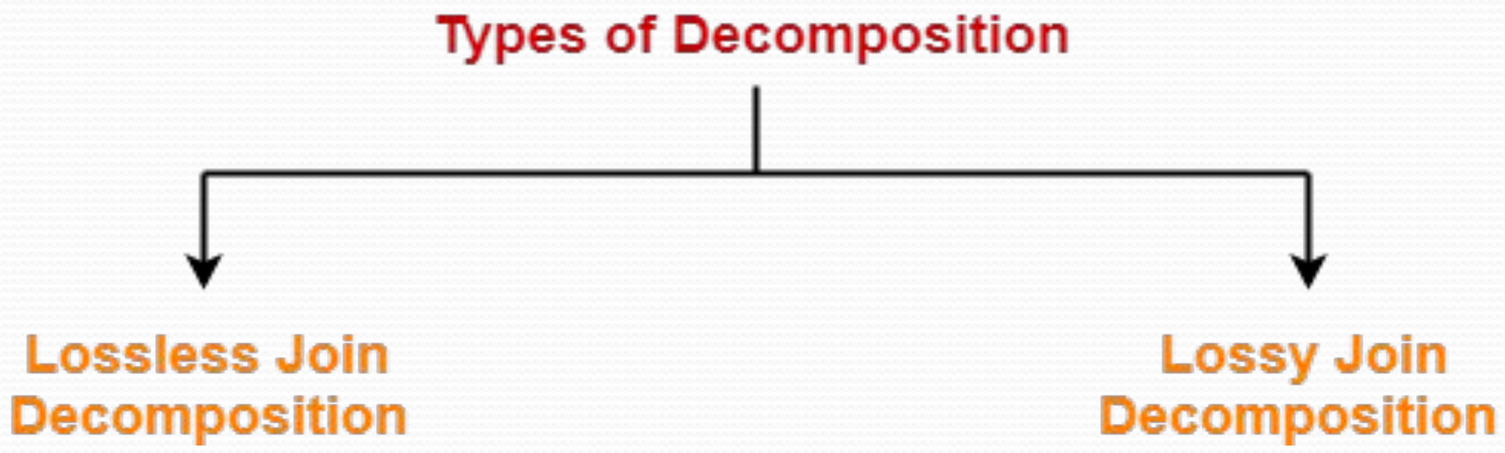
R2

Geeks	geeks
a1	c1
a2	c2

R

Geeks	for	geeks
a1	b1	c1
a2	b1	c2

- Decomposition of a relation can be completed in the following two ways-



Determining Whether Decomposition Is Lossless Or Lossy

Consider a relation R is decomposed into two sub relations R_1 and R_2.

Then,

If all the following conditions satisfy, then the decomposition is lossless.

If any of these conditions fail, then the decomposition is lossy.

Condition-1:

Union of both the sub relations must contain all the attributes that are present in the original relation R. Thus, $\mathbf{R_1 \cup R_2 = R}$

Condition-2:

Intersection of both the sub relations must not be null.

In other words, there must be some common attribute which is present in both the sub relations.

Thus, $\mathbf{R1 \cap R2 \neq \varnothing}$

Condition-3:

Intersection of both the sub relations must be a super key of either R_1 or R_2 or both. Thus, $\mathbf{R_1 \cap R_2}$ **= Super key of $\mathbf{R_1}$ or $\mathbf{R_2}$**

Lossless Decomposition

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Lossy Decomposition

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Canonical Cover / Minimal cover/ Irreducible set of FD

- A canonical cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, it is also called as **Irreducible set**.

Characteristics-

- Canonical cover is free from all the extraneous functional dependencies.
- The closure of canonical cover is same as that of the given set of functional dependencies.
- Canonical cover is not unique and may be more than one for a given set of functional dependencies.
- **Extraneous attributes:** An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

Lets say F’ is a functional dependency then F’ is a canonical cover

If F’ does not have

1. Extraneous attributes / Redundant attributes
2. Redundant FD

Steps:

2. Splitting rule so that in every FD right hand side has single attributes
3. Remove extraneous attributes
4. Remove Redundant FD

Example

F={AB →C,………..,A→C}

Here A alone can determine C so B is extraneous attribute

Example

F={AB → C,C →AB,B →C,ABC →AC,A →C,AC →B}

Making single dependent on right hand side

Step 1: = {AB → C,C →A, C → B,B →C,~~ABC →A~~, ~~ABC →C~~,A →C,AC →B}

Discarding trivial function

Step 2: {AB → C,C →A, C → B,B →C,A →C,AC →B}

Removing redundant FD

Step 3: = {C →A, C → B, B →C, A →C}Canonical cover

Normalization of Database

- Database Normalization is a technique of organizing the data in the database.
- Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- **Normalization is used for mainly two purposes:**
  - Eliminating redundant(useless) data.
  - Ensuring data dependencies make sense i.e data is logically stored.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are :

- •Insertion anomaly
- •Update anomaly
- •Deletion anomaly

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Sabin	Pulchowk	D001
101	Sabin	Pulchowk	D002
123	Mohan	New Road	D890
166	Rabin	Kalimati	D900
166	Rabin	Kalimati	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Sabin as he belongs to two departments of the company. If we want to update the address of Sabin then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Sabin would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

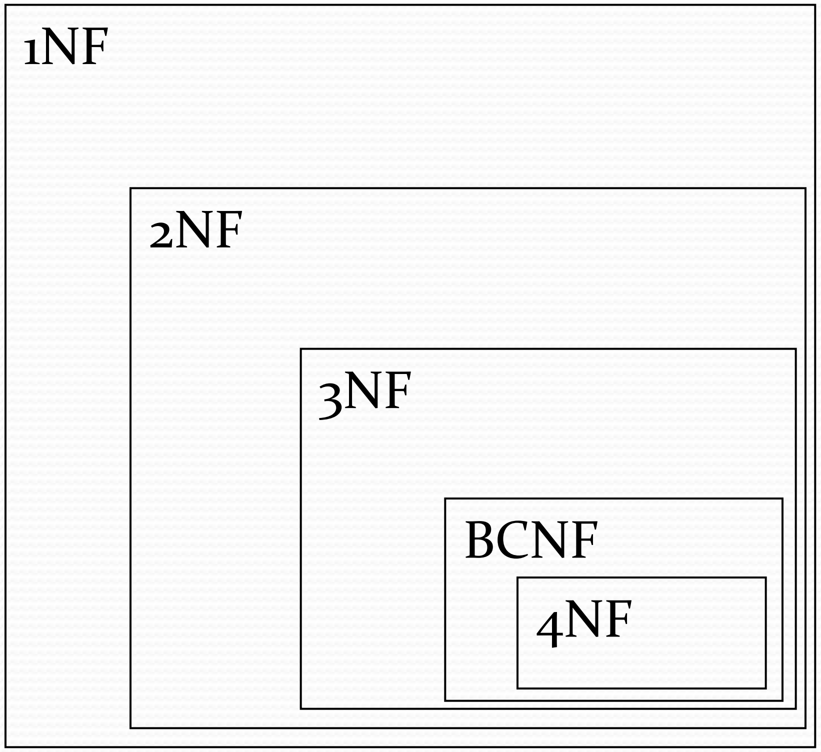
Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Mohan since he is assigned only to this department.

Normalization Rule

Normalization rules are divided into the following normal forms:

- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Fourth Normal Form

Normalization



a relation in 4NF, is also in BCNF

a relation in BCNF, is also in 3NF

a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

- It should only have single(atomic) valued attributes/columns.
- Values stored in a column should be of the same domain
- All the columns in a table should have unique names.
- And the order in which data is stored, does not matter.

Example

roll_no	name	subject
101	Ram	DBMS, C
103	Shyam	Java
102	Sita	C, C++

As per the 1st Normal form each column must contain atomic value.

Above table is not in first normal form
How to solve this Problem?

roll_no	name	subject
101	Ram	DBMS
101	Ram	C
103	Shyam	Java
102	Sita	C
102	Sita	C++

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row. Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

- It should be in the First Normal form.
- And, all the non prime key should be fully functional depend on candidate key.

Customer_id	Store_id	location
1	1	Pulchowk
1	3	Kalimati
2	1	Pulchowk
3	2	New Road
4	3	Kalimati

Customer_id	Store_id	location
1	1	Pulchowk
1	3	Kalimati
2	1	Pulchowk
3	2	New Road
4	3	Kalimati

Candidate key : customer_id , store_id
Prime attributes: customer_id , store_id
Non prime attribute: Location

Customer_id	Store_id
1	1
1	3
2	1
3	2
4	3

Store_id	location
1	Pulchowk
2	New Road
3	Kalimati

3rd Normal Form Definition

A database is in third normal form if it satisfies the following conditions:

- It is in second normal form
- There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

Consider the following example:

Book_ID	Genre_ID	GenreType	Price
1	1	Gardening	25
2	2	Sports	20
3	1	Gardening	25
4	3	Travel	15
5	2	Sports	20

In the table able, [Book_ID] determines [Genre_ID], and [Genre_ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

Book

Book_ID	Genre_ID	Price
1	1	25
2	2	20
3	1	25
4	3	15
5	2	20

Genre

Genre_ID	GenreType
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [GENRE], [Genre Type] is only dependent on [Genre ID].

Boyce-Codd Normal Form or BCNF

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Employee table

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	Nepal	Designing	D394	283
264	Nepal	Testing	D394	300
364	USA	Stores	D283	232
364	USA	Developing	D283	549

Functional dependencies

Emp_ID → Nationality

Emp_Dept → {Dept_Type, Dept_No}

Candidate key

{Emp_ID, Emp_Dept}

In this example, the table is not in BCNF form as both the Emp_ID and Emp_Dept alone are not keys. To convert the table into BCNF form, decompose the table into three tables based on the functional dependency.

- **EMP_COUNTRY**

EMP_ID	EMP_COUNTRY
264	Nepal
364	USA

- **Dept table**

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies

Emp_ID → Nationality

Emp_Dept → {Dept_Type, Dept_No}

Candidate key

Nationality Table: Emp_ID

Dept Table: Emp_Dept

Dept Mapping Table: {Emp_ID, Emp_Dept}

The relation is now in BCNF form because it satisfies both conditions which are that the table is already in 3NF form and on the LHS of the functional dependency there is a candidate key.

4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

- It should be in the **Boyce-Codd Normal Form**.
- And, the table should not have any **Multi-valued Dependency**.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

In this example, course and hobby are independent of each other but dependent on s_id. In this example, these two columns are said to be multivalue dependent on s_id.

To make the above relation satisfy the 4th normal form, we can decompose the table into 2 tables.

Course

s_id	course
1	Science
1	Maths
2	C#
2	Php

hobby

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form.