

Functional Dependency

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.

For example:

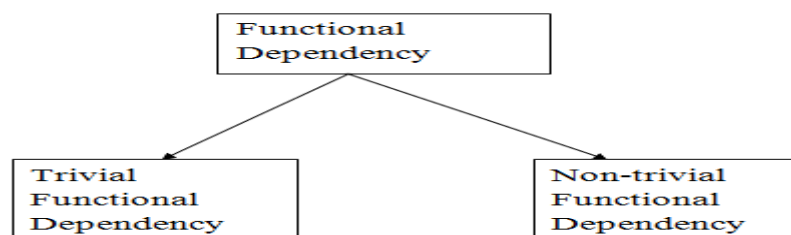
Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address. Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

$\text{Emp_Id} \rightarrow \text{Emp_Name}$

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Example:

Consider a table with two columns Employee_Id and Employee_Name.

$\{\text{Employee_id}, \text{Employee_Name}\} \rightarrow \text{Employee_Id}$ is a trivial functional dependency as Employee_Id is a subset of $\{\text{Employee_Id}, \text{Employee_Name}\}$.

Also, $\text{Employee_Id} \rightarrow \text{Employee_Id}$ and $\text{Employee_Name} \rightarrow \text{Employee_Name}$ are trivial dependencies too.

2. Non-trivial functional dependency

- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
- When $A \cap B$ is NULL, then $A \rightarrow B$ is called as complete non-trivial.

Example:

$\text{ID} \rightarrow \text{Name}$,

$\text{Name} \rightarrow \text{DOB}$

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:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

If $X \supseteq Y$ then $X \rightarrow Y$

Example:

$X = \{a, b, c, d, e\}$

$Y = \{a, b, c\}$

2. 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 $XZ \rightarrow YZ$

Example:

For $R(ABCD)$, if $A \rightarrow B$ then $AC \rightarrow BC$

3. 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 $X \rightarrow Z$

4. 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 $X \rightarrow YZ$

Proof:

$X \rightarrow Y$ (given)

$X \rightarrow Z$ (given)

$X \rightarrow XY$ (using IR_2 on 1 by augmentation with X. Where $XX = X$)

$XY \rightarrow YZ$ (using IR_2 on 2 by augmentation with Y)

$X \rightarrow YZ$ (using IR_3 on 3 and 4)

5. 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 $X \rightarrow Z$

Proof:

$X \rightarrow YZ$ (given)

$YZ \rightarrow Y$ (using IR_1 Rule)

$X \rightarrow Y$ (using IR_3 on 1 and 2)

6. 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 $YZ \rightarrow W$ then $XZ \rightarrow W$

Proof:

1. $X \rightarrow Y$ (given)

2. $WY \rightarrow Z$ (given)

3. $WX \rightarrow WY$ (using IR_2 on 1 by augmenting with W)

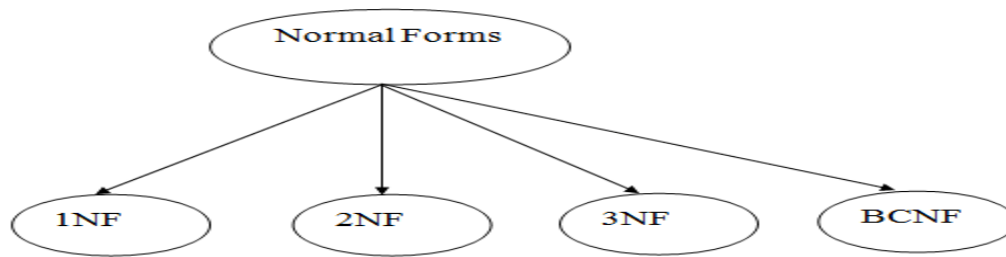
4. $WX \rightarrow Z$ (using IR_3 on 3 and 2)

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:



The database normalization process is divided into following the normal form:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

Normal Form	Description
<u>1NF</u>	A relation is in 1NF if it contains an atomic value.
<u>2NF</u>	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
<u>3NF</u>	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
<u>4NF</u>	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
<u>5NF</u>	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example-1: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Example-2 :

Sample Employee table, it displays employees are working with multiple departments.

Employee	Age	Department
Melvin	32	Marketing, Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

Employee table following 1NF:

Employee	Age	Department
Melvin	32	Marketing
Melvin	32	Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30

25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Sample Products table:

productID	product	Brand
1	Monitor	Apple
2	Monitor	Samsung
3	Scanner	HP
4	Head phone	JBL

Product table following 2NF:

Products Category table:

productID	product
1	Monitor
2	Scanner

3	Head phone
---	------------

Brand table:

brandID	brand
1	Apple
2	Samsung
3	HP
4	JBL

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

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

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

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

In the above table Functional dependencies are as follows:

1. $EMP_ID \rightarrow EMP_COUNTRY$
2. $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_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:

1. $EMP_ID \rightarrow EMP_COUNTRY$
2. $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

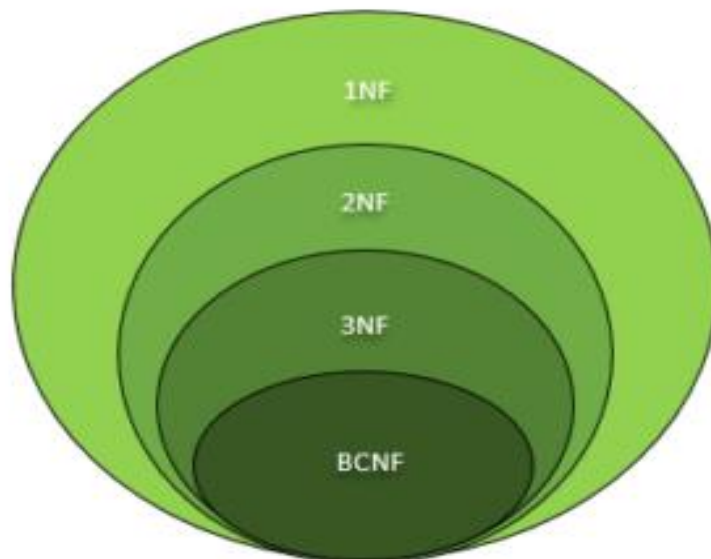
Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.



Multivalued Dependency (MVD)

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL $\twoheadrightarrow$ MANUF_YEAR
2. BIKE_MODEL $\twoheadrightarrow$ COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

When existence of one or more rows in a table implies one or more other rows in the same table, then the Multi-valued dependencies occur.

If a table has attributes P, Q and R, then Q and R are multi-valued facts of P. It is represented by double arrow –

$\twoheadrightarrow$

For our example:

P $\twoheadrightarrow$ Q
P $\twoheadrightarrow$ R

In the above case, Multivalued Dependency exists only if Q and R are independent attributes. A table with multivalued dependency violates the 4NF.

Example

Let us see an example &min;

<Student>

StudentName	CourseDiscipline	Activities
Amit	Mathematics	Singing
Amit	Mathematics	Dancing
Yuvraj	Computers	Cricket
Akash	Literature	Dancing
Akash	Literature	Cricket
Akash	Literature	Singing

In the above table, we can see Students **Amit** and **Akash** have interest in more than one activity.

This is multivalued dependency because **CourseDiscipline** of a student are independent of Activities, but are dependent on the student.

Therefore, multivalued dependency –

StudentName ->-> CourseDiscipline
StudentName ->-> Activities

The above relation violates Fourth Normal Form in Normalization.

To correct it, divide the table into two separate tables and break Multivalued Dependency –

<StudentCourse>

StudentName	CourseDiscipline
Amit	Mathematics
Amit	Mathematics
Yuvraj	Computers
Akash	Literature
Akash	Literature
Akash	Literature

<StudentActivities>

StudentName	Activities
Amit	Singing
Amit	Dancing
Yuvraj	Cricket
Akash	Dancing
Akash	Cricket

Akash	Singing
-------	---------

This breaks the multivalued dependency and now we have two functional dependencies –

StudentName -> CourseDiscipline
StudentName -> Activities

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**.

So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Decomposition of a Relation-

The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

Properties of Decomposition-

The following two properties must be followed when decomposing a given relation-

1. Lossless decomposition-

Lossless decomposition ensures-

- No information is lost from the original relation during decomposition.
- When the sub relations are joined back, the same relation is obtained that was decomposed.

Every decomposition must always be lossless.

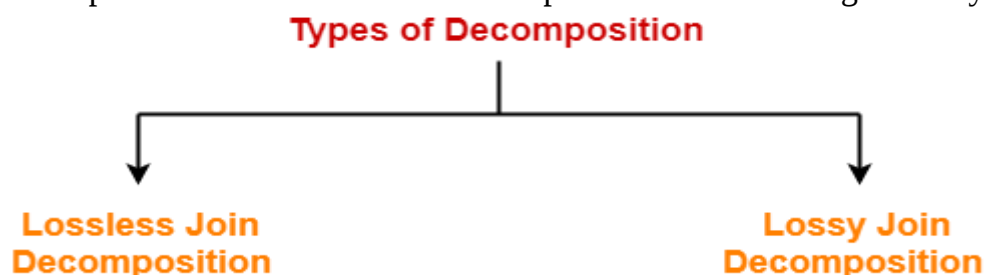
2. Dependency Preservation-

Dependency preservation ensures-

- None of the functional dependencies that holds on the original relation are lost.
- The sub relations still hold or satisfy the functional dependencies of the original relation.

Types of Decomposition-

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



1. Lossless Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations $R_1, R_2, \dots, R_n$.
- This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
- For lossless join decomposition, we always have-

$$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n = R$$

where $\bowtie$ is a natural join operator

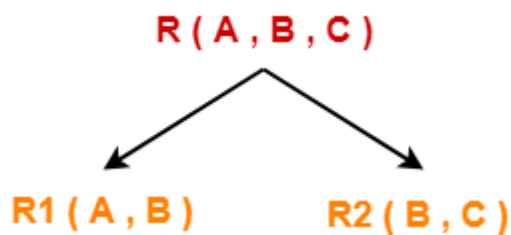
Example-

Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations R₁(A , B) and R₂(B , C)-



The two sub relations are-

A	B
1	2
2	5
3	3

R₁(A , B)

B	C
2	1
5	3
3	3

R₂(B , C)

Now, let us check whether this decomposition is lossless or not.

For lossless decomposition, we must have- **R₁ $\bowtie$ R₂ = R**

Now, if we perform the natural join ($\bowtie$) of the sub relations R₁ and R₂ , we get-

A	B	C
---	---	---

1	2	1
2	5	3
3	3	3

This relation is same as the original relation R.

Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE-

Lossless join decomposition is also known as **non-additive join decomposition**.

- This is because the resultant relation after joining the sub relations is same as the decomposed relation.
- No extraneous tuples appear after joining of the sub-relations.

Let us see an example:

<EmpInfo>

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:

<EmpDetails>

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

<DeptDetails>

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

Now, Natural Join is applied on the above two tables:

The result will be:

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

Therefore, the above relation had lossless decomposition i.e. no loss of information.

2. Lossy Join Decomposition-

Consider there is a relation R which is decomposed into sub relations $R_1, R_2, \dots, R_n$.

- This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
- The natural join of the sub relations is always found to have some extraneous tuples.
- For lossy join decomposition, we always have-

$$R_1 \bowtie R_2 \bowtie R_3 \dots \dots \bowtie R_n \supset R$$

where $\bowtie$ is a natural join operator

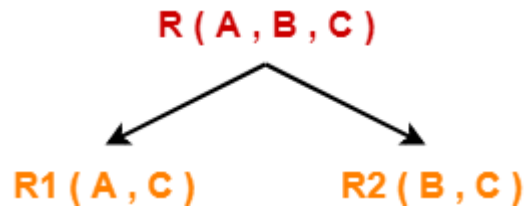
Example-

Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations as $R_1(A , C)$ and $R_2(B , C)$ -



The two sub relations are-

A	C
1	1
2	3
3	3

R₁(A , B)

B	C
2	1
5	3
3	3

R₂(B , C)

Now, let us check whether this decomposition is lossy or not.

For lossy decomposition, we must have-

$$R_1 \bowtie R_2 \supset R$$

Now, if we perform the natural join ($\bowtie$) of the sub relations R_1 and R_2 we get-

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not same as the original relation R and contains some extraneous tuples.

Clearly, $R_1 \bowtie R_2 \supset R$.

Thus, we conclude that the above decomposition is lossy join decomposition.

NOTE-

Lossy join decomposition is also known as **careless decomposition**.

- This is because extraneous tuples get introduced in the natural join of the sub-relations.
- Extraneous tuples make the identification of the original tuples difficult.