

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.

$$1. X \rightarrow Y$$

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.

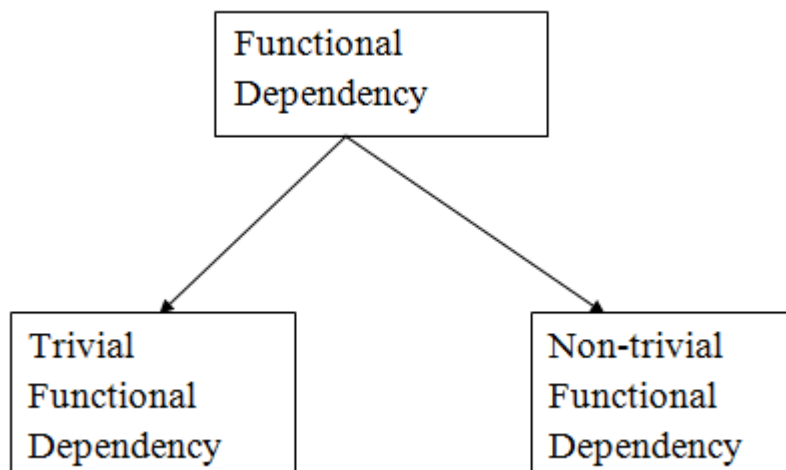
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

$$1. \text{Emp_Id} \rightarrow \text{Emp_Name}$$

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:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. $\{\text{Employee_id}, \text{Employee_Name}\} \rightarrow \text{Employee_Id}$ is a trivial functional dependency as
3. Employee_Id is a subset of $\{\text{Employee_Id}, \text{Employee_Name}\}$.
4. 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:

1. $\text{ID} \rightarrow \text{Name}$,
2. $\text{Name} \rightarrow \text{DOB}$

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is 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 undesirable characteristics like Insertion, Update, and Deletion Anomalies.

- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Following are the various types of Normal forms:

| | 1NF | 2NF | 3NF | 4NF | 5NF |
|---------------------------|----------------------------|---|---|--|---|
| Decomposition of Relation | R | R ₁₁ R ₁₂ | R ₂₁ R ₂₂ R ₂₃ | R ₃₁ R ₃₂ R ₃₃ R ₃₄ | R ₄₁ R ₄₂ R ₄₃ R ₄₄ R ₄₅ |
| Conditions | Eliminate Repeating Groups | Eliminate Partial Functional Dependency | Eliminate Transitive Dependency | Eliminate Multi-values Dependency | Eliminate Join Dependency |

| Normal Form | Description |
|-------------|--|
| 1NF | A relation is in 1NF if it contains an atomic value. |

| | |
|------|---|
| 2NF | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| 3NF | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| 4NF | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| 5NF | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

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: 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 |

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 |

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_firsttable: 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.

Inclusion Dependenc

- Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
- Inclusion dependencies are quite common. They typically show little influence on designing of the database.
- The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
- The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
- Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.
- Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
- In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.
- In practice, most inclusion dependencies are key-based that is involved only keys.

Lossless-join

Lossless-join decomposition is a process in which a relation is decomposed into two or more relations. This property guarantees that the extra or less tuple generation problem does not occur and no information is lost from the original relation during the decomposition. It is also known as non-additive join decomposition.

When the sub relations combine again then the new relation must be the same as the original relation was before decomposition.

Consider a relation R if we decomposed it into sub-parts relation R1 and relation R2.

The decomposition is lossless when it satisfies the following statement –

- If we union the sub Relation R1 and R2 then it must contain all the attributes that are available in the original relation R before decomposition.
- Intersections of R1 and R2 cannot be Null. The sub relation must contain a common attribute. The common attribute must contain unique data.

The common attribute must be a super key of sub relations either R1 or R2.

Here,

$R = (A, B, C)$

$R1 = (A, B)$

$R2 = (B, C)$

The relation R has three attributes A, B, and C. The relation R is decomposed into two relation R1 and R2. . R1 and R2 both have 2-2 attributes. The common attributes are B.

The Value in Column B must be unique. if it contains a duplicate value then the Lossless-join decomposition is not possible.

Draw a table of Relation R with Raw Data –

R (A, B, C)

| A | B | C |
|----|----|----|
| 12 | 25 | 34 |
| 10 | 36 | 09 |
| 12 | 42 | 30 |

It decomposes into the two sub relations –

R1 (A, B)

| A | B |
|----|----|
| 12 | 25 |
| 10 | 36 |
| 12 | 42 |

R2 (B, C)

| B | C |
|----|----|
| 25 | 34 |

| B | C |
|----------|----------|
| 36 | 09 |
| 42 | 30 |

Now, we can check the first condition for Lossless-join decomposition.

The union of sub relation R1 and R2 is the same as relation R.

$R_1 \cup R_2 = R$

We get the following result –

| A | B | C |
|----------|----------|----------|
| 12 | 25 | 34 |
| 10 | 36 | 09 |
| 12 | 42 | 30 |

The relation is the same as the original relation R. Hence, the above decomposition is Lossless-join decomposition.

Multivalued Dependency

- 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".

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD $\bowtie \{R1, R2, \dots, Rn\}$ is said to hold over a relation R if R1, R2, ..., Rn is a lossless-join decomposition.
- The $\pi(A, B, C, D), \pi(C, D)$ will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, $\pi(R1, R2, R3)$ is used to indicate that relation R1, R2, R3 and so on are a JD of R.