Functional Dependency in DBMS

In a database, just storing information is not enough. It must be well-arranged, without repeating values, and consistently accurate. Functional dependency helps with this by showing how one value in a table depends on another. Reasoning about functional dependencies in DBMS helps database designers logically understand data relationships, which is crucial for creating efficient schemas. It explains how one part of the data is linked to another, which supports proper design and a clean structure. In this blog, you will learn what functional dependency is in DBMS, why it is useful, and how its different types help in making a good database design.

Table of Contents:

What is Functional Dependency in DBMS?

Functional Dependency in DBMS is a concept in relational databases. It describes how one piece of data relates to another. That is, if knowing the value of one column (or set of columns) helps us find the value of another column, then there is functional dependency.

Usually written as:

A -> B, meaning "A determines B"

Explanation: Here, knowing the value of A lets us find exactly one value of B.

Let us understand this with an example:

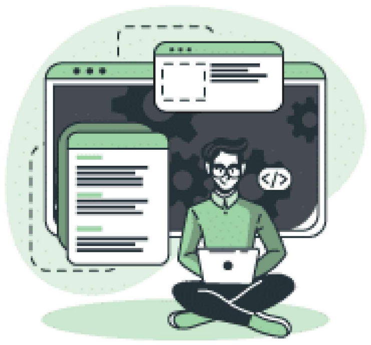
Consider the following student table:

StudentID	Name	Email
101	Yash	Yash@email.com
102	Ishan	Ishu@email.com

In this case:

- StudentID -> Name
- StudentID -> Email

Explanation: Here, this dependency shows that with the help of **studentID,** we can find the name and email of the students.



Advantages of Functional Dependency in DBMS

1. Facilitates Normalization: Functional dependency helps to arrange data better by breaking a large table into smaller ones using related attributes and removing repeated values.

2. Eliminates Redundancy: Functional dependency helps in understanding how one set of data depends on another, which avoids storing the same information more than once.

3. Improves Accuracy: Functional dependency reduces redundancy and supports normalization, which makes the data more accurate and consistent.

4. Easier to Manage: Functional dependency makes it easier to manage and update the database since the data is well-linked and structured.

5. Better Query Performance: Functional dependency creates organized tables, which makes searching and updating faster and improves system performance.

Types of Functional Dependency in DBMS

Here are the main types of functional dependencies in DBMS used to organize and manage data in a database.

1. Trivial Functional Dependency

A functional dependency is called trivial when the attribute on the right side is already part of the left side. This means it does not give any new information about the data. These types of dependencies are always true, but are not very useful in database design.

Example: {StudentID, Name} ->StudentID

2. Non-Trivial Functional Dependency

A functional dependency is non-trivial when the right side is not included in the left side. It shows a real link between two sets of data and helps decide how data should be split into different tables during normalization.

Example: StudentID –> Name

This means that the Name can be found using the StudentID. But Name is not included in StudentID, so this is a non-trivial dependency.

3. Semi-Non-Trivial Functional Dependency

A semi-non-trivial functional dependency is a mix of both trivial and non-trivial dependencies. In this case, part of the right side is already present on the left side, but it also has extra attributes that are not on the left side. This makes it only partly useful and still important to consider in database design.

Note: This is a custom or non-standard term used to describe a case that includes both trivial and non-trivial parts. It is not usually found in official database books or academic material.

Example: {StudentID, Name} **->** Name, Email

In this case, Name is present on the left side (trivial), while Email is present on the right side (non-trivial).

4. Multivalued Functional Dependency

In this case, one attribute is linked to many values in another, even without depending on other columns. This shows that an entity can hold multiple values for the same attribute. The fourth normal form helps fix this by removing repeated or unnecessary data.

Example: StudentID **->**-> PhoneNumber

Every student can have more than one phone number, even though they are all connected to the same student.

5. Transitive Functional Dependency

A transitive dependency happens when one attribute depends on another through a third attribute. It is like an indirect link and can lead to repeated or extra data. Transitive dependencies are removed when applying the third normal form.

Example: StudentID **->** DepartmentID and DepartmentID **->** DepartmentName

Since StudentID leads to DepartmentID and DepartmentID leads to DepartmentName, **therefore, StudentID -> DepartmentName is a transitive dependency**.

6. Fully Functional Dependency

A dependency is fully functional if the attribute is dependent on the **entire** composite key and **not** on a part of it. It prevents partial dependencies from being used, which is important for the Second Normal Form (2NF). If it were just depending on one half of the composite key, then it would be a partial dependency.

Example: (StudentID, CourseID) **->** Marks

Here, the marks depend on both the **StudentID and CourseID** together, not on just one of them.

Properties of Functional Dependency in DBMS

There are a few fundamental rules, or more specifically, properties of functional dependencies. These are called Armstrong's Axioms, and they allow us to find new FDs from the FDs we currently have.

1. Reflexivity

If an attribute set B is a subset of A, then A -> B holds true.

Example: {StudentID, Name} -> StudentID

2. Augmentation

If A -> B holds, then adding the same attribute to both sides also holds: AC -> BC.

Example: If StudentID -> Name, then StudentID and CourseID -> Name and CourseID.

3. Transitivity

If A -> B and B -> C, then A -> C.

This means if one attribute depends on a second, and the second depends on a third, then the first indirectly determines the third.

Example: StudentID -> DeptID, DeptID -> DeptName then StudentID -> DeptName

These rules help us recognize, simplify, and predict new dependencies that can be utilized with normalization and building queries.

How to Identify Functional Dependencies in DBMS Tables

Identifying functional dependencies within a table can be as simple as identifying unique values and their relationships. Reasoning about functional dependencies in DBMS involves analyzing how one attribute determines another based on these patterns.

Steps to Follow:

Step 1: Examine the Primary Key

A primary key consists of a unique column (or set of columns) that identifies each row in a table. By default, the primary key will determine all of the other columns in that row.

Example: Students Table

StudentID	Name	Email
101	John	john@email.com
102	Sarah	sarah@email.com

As we can see, **StudentID** is acting as the primary key. So we can say:

- **StudentID -> Name**
- **StudentID -> Email**

Step 2: Look for Repeating Values

Check if the value in one column repeats in multiple rows or not, and if it links to the same value in another column.

Example: Departments Table:

Department	DeptLocation
IT	Building A
HR	Building B
IT	Building A

As we can see in the above-mentioned example, **"IT"** appears twice, and both are linked to **"Building A"** every time.

At this stage, **reasoning about functional dependencies in DBMS** becomes useful because it relies on critical thinking to analyze the links between attributes

Step 3: Use Logical Thinking

Ask one question while analyzing your data. If you know the value of one column, you will be able to find out the value of another column.

Example:

Department	DeptLocation
IT	Building A
HR	Building B
IT	Building A

Now let's see the following example:

- **EmployeeID -> Name**: Yes, because each EmployeeID is unique and has only one Name.
- **Name -> Department**: No, because the same Name (like Ramesh) appears multiple times and may belong to different departments.

Therefore, EmployeeID **->** Name is a proper functional dependency, but Name **->** Department is not a functional dependency.

Role of Functional Dependency in DBMS Normalization and Database Design

Normalization is the process of organizing tables to eliminate redundant or unnecessary data. Functional Dependency is at the heart of this process, guiding how attributes are grouped and separated across tables.

Here's how functional dependencies relate to different normal forms:

- **1NF (First Normal Form):** Remove repeating groups and ensure each column contains atomic (indivisible) values.
- **2NF (Second Normal Form):** Eliminate partial dependencies — i.e., when a non-prime attribute depends only on a part of a composite primary key. This is where identifying **fully functional dependencies in DBMS** becomes important.
- **3NF (Third Normal Form):** Remove transitive dependencies — when one attribute depends on another through a third one.
- **BCNF (Boyce-Codd Normal Form):** A stricter version of 3NF where the left side of every functional dependency must be a superkey (i.e., must uniquely identify each row).

Beyond normalization, functional dependencies guide the overall structure and efficiency of database design. They help organize data and ensure integrity.

A well-designed database using functional dependencies will:

- Reduce unnecessary data repetitions across tables
- Maintain consistency through uniform data formats
- Simplify updates and modifications
- Improve performance and scalability

Designers rely on functional dependencies to:

- Determine which attributes belong in which table
- Split larger tables into smaller, related ones (decomposition)
- Avoid breaking normalization rules during schema design

Best Practices for Using Functional Dependency in DBMS

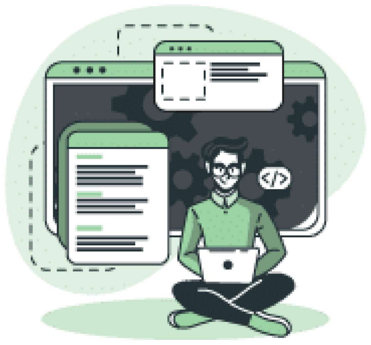
1. Identify keys first: You will want to begin by identifying primary and candidate keys. Most FDs will flow from these.

2. Remove partial and transitive dependencies: This is what leads to data redundancy, and you will want to eliminate these toward 2NF and 3NF.

3. Use decomposition when needed: Be cautious when breaking tables apart to ensure data is not lost and dependencies are preserved.

4. Check dependency preservation: All original FDs should still be valid after the decomposition.

5. Maintain simplicity and readability in table design: Do not make your schema overly complex by including too many redundant dependencies.
Kickstart Your SQL Journey – For Free!
Learn how to write powerful queries and manage databases with our beginner-friendly free SQL course.
Explore Program



Conclusion

Functional dependency is important in database design. It shows how one column depends on another and helps break large tables into smaller ones. This reduces repeated data, keeps information accurate, and makes the database easier to use. A well-designed database using functional dependencies is faster, more reliable, and easier to manage in the long run. This results in better performance, consistent data, and improved data integrity.

Take your skills to the next level by enrolling in the SQL Course today and gaining hands-on experience. Also, prepare for job interviews with SQL Interview Questions prepared by industry experts.

Check out the other blogs related to SQL:

Functional Dependency in DBMS – FAQs

Q1. What is functional dependency in DBMS?

Functional dependency in DBMS refers to the relationship between two attributes, typically between a key and a non-key attribute. If knowing the value of attribute A allows you to determine the value of attribute B, we say B is functionally dependent on A (written as A → B).
Q2. What is the role of functional dependency in DBMS normalization?

Functional dependency is the foundation of normalization. It helps identify redundant data and guides how to split tables into smaller, logically related tables to reduce anomalies and maintain data consistency.
Q3. What are the types of functional dependency in DBMS?

The main types of functional dependencies in DBMS include:

Trivial functional dependency:
Non-trivial functional dependency
Semi-non-trivial functional dependency
Multivalued functional dependency
Transtive functional dependency
Fully functional dependency
Q4. What is trivial functional dependency in DBMS?

A trivial functional dependency occurs when the right-hand side attribute is already part of the left-hand side. For example, in {StudentID, Name} → StudentID, the dependency is trivial because StudentID is already included in the left side.
Q5. What is fully functional dependency in DBMS with example?

A fully functional dependency in DBMS occurs when an attribute is dependent on the entire composite key, not just a part of it.
Example: (StudentID, CourseID) → Marks
Here, Marks depends on both StudentID and CourseID together, not on either one alone.

Q6. How does reasoning about functional dependencies in DBMS help in normalization?

Reasoning about functional dependencies in DBMS allows database designers to logically analyze how data attributes relate to one another. This helps ensure correct decomposition, removes redundancy, and supports the creation of normalized tables that are efficient and consistent.