**What is Functional Dependency**

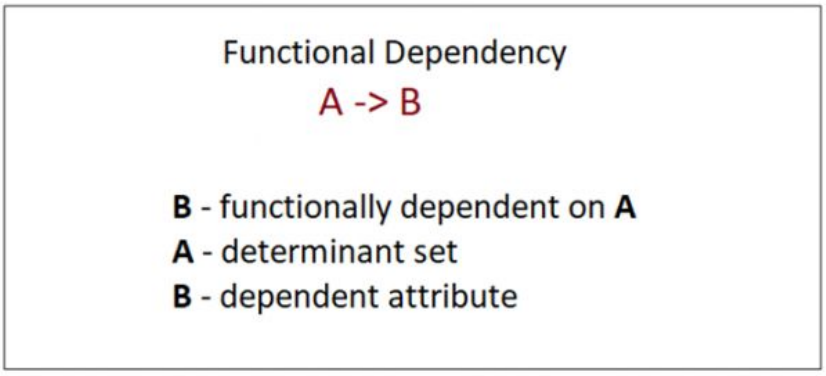
Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.

To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)

Then the following will represent the functional dependency between attributes with an arrow sign −

| **A -> B** |
|---|

Above suggests the following:




**Example**

The following is an example that would make it easier to understand functional dependency −

We have a **<Department>** table with two attributes − **DeptId** and **DeptName**.

| **DeptId** = Department ID<br>**DeptName** = Department Name |
|---|

The **DeptId** is our primary key. Here, **DeptId** uniquely identifies the **DeptName** attribute. This is because if you want to know the department name, then at first you need to have the **DeptId**.

| **DeptId** | **DeptName** |
|---|---|
| 001 | Finance |
| 002 | Marketing |
| 003 | HR |

Therefore, the above functional dependency between **DeptId** and **DeptName** can be determined as **DeptId** is functionally dependent on **DeptName** −

```
DeptId -> DeptName
```

**Types of Functional Dependency**

Functional Dependency has three forms −

- Trivial Functional Dependency
- Non-Trivial Functional Dependency

Let us begin with Trivial Functional Dependency −

**Trivial Functional Dependency**

It occurs when B is a subset of A in −

```
A ->B
```

**Example**

We are considering the same **<Department>** table with two attributes to understand the concept of trivial dependency.

The following is a trivial functional dependency since **DeptId** is a subset of **DeptId** and **DeptName**

```
{ DeptId,  DeptName } -> Dept Id
```

**Non –Trivial Functional Dependency**

It occurs when B is not a subset of A in −

```
A ->B
```

**Example**

```
DeptId ->  DeptName
```

The above is a non-trivial functional dependency since DeptName is not a subset of DeptId.

**Armstrong's Axioms Property of Functional Dependency**

Armstrong's Axioms property was developed by William Armstrong in 1974 to reason about functional dependencies.

The property suggests rules that hold true if the following are satisfied:

- **Transitivity**
  If A->B and B->C, then A->C i.e. a transitive relation.
- **Reflexivity**
  A-> B, if B is a subset of A.
- **Augmentation**
  The last rule suggests: AC->BC, if A->B

**Advantages of Functional Dependency**

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

**Summary**

- Functional Dependency is when one attribute determines another attribute in a DBMS system.
- Axiom, Decomposition, Dependent, Determinant, Union are key terms for functional dependency
- Four types of functional dependency are 1) Multivalued 2) Trivial 3) Non-trivial 4) Transitive
- Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table
- The Trivial dependency occurs when a set of attributes which are called a trivial if the set of attributes are included in that attribute
- Nontrivial dependency occurs when A->B holds true where B is not a subset of A
- A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies
- Normalization is a method of organizing the data in the database which helps you to avoid data redundancy

**What is Transitive Dependency**

When an indirect relationship causes functional dependency it is called Transitive Dependency.

If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

To achieve 3NF, eliminate the Transitive Dependency.

**Example**

**<MovieListing>**

| **Movie_ID** | **Listing_ID** | **Listing_Type** | **DVD_Price ($)** |
|---|---|---|---|
| M08 | L09 | Crime | 180 |
| M03 | L05 | Drama | 250 |
| M05 | L09 | Crime | 180 |

The above table is not in 3NF because it has a transitive functional dependency −

| **Movie_ID -> Listing_ID**<br>**Listing_ID -> Listing_Type** |
|---|

Therefore, the following has transitive functional dependency.

**Movie_ID -> Listing_Type**

The above states the relation <MovieListing> violates the 3rd Normal Form (3NF).

To remove the violation, you need to split the tables and remove the transitive functional dependency.

**<Movie>**

**Movie_ID**

| **Listing_ID** | **DVD_Price ($)** | |
|---|---|---|
| M08 | L09 | 180 |
| M03 | L05 | 250 |
| M05 | L09 | 180 |

**<Listing>**

| **Listing_ID** | **Listing_Type** |
|---|---|
| L09 | Crime |
| L05 | Drama |
| L09 | Crime |

**What is Partial Dependency?**

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

The 2nd Normal Form (2NF) eliminates the Partial Dependency.

Let us see an example −

**Example**

**<StudentProject>**

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

In the above table, we have partial dependency; let us see how −

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

| |
|---|
| **StudentID** = Unique ID of the student<br>**StudentName** = Name of the student<br>**ProjectNo** = Unique ID of the project<br>**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 |

## Fully-functional dependency in DBMS

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

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

Let us see an example −

**<ProjectCost>**

| **ProjectID** | **ProjectCost** |
|---|---|
| 001 | 1000 |
| 001 | 5000 |

**<EmployeeProject>**

| **EmpID** | **ProjectID** | **Days** |
|---|---|---|
| E099 | 001 | 320 |
| E056 | 002 | 190 |

The above relations states that −

Days are the number of days spent on the project.

| **EmpID, ProjectID, ProjectCost -> Days** |
|---|

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the {Days} spent on the project by the employee.

This summarizes and gives our fully functional dependency −

| **{EmpID, ProjectID} -> (Days)** |
|---|