Functional Dependency

Overview

Functional Dependency is the relationship between attributes(*characteristics*) of a table related to each other.

The functional dependency of A on B is represented by **A → B**,
where **A** and **B** are the attributes of the relation.

What is Functional Dependency in DBMS?

- Relational database is a **collection of data** stored in rows and columns.
- Columns represent the *characteristic* of data while each row in a table represents a set of *related data*, and every row in the table has the same structure.
- The row is sometimes referred to as a **tuple in DBMS**.

- **Functional Dependency**, as the name suggests it is the relationship between attributes(*characteristics*) of a table related to each other.
- A relation consisting of functional dependencies always follows a set of rules called *RAT* rules.
- They are proposed by **William Armstrong** in 1974.
- It helps in maintaining the **quality of data** in the database, and the core concepts behind database normalization are based on functional dependencies.

How to Denote a Functional Dependency:

- A functional dependency is denoted by an arrow "→".
- The functional dependency of **A** on **B** is represented by A → B.
- Sometimes everything on the left side of functional dependency is also referred to as **determinant set**, while everything on the right side is referred to as **depending attributes**.
- Pointing arrows determines the depending attribute and the origin of the arrow determines the determinant set.

Types of FD:

1. **Trivial functional dependency**
2. **Non-Trivial functional dependency**
3. **Multivalued functional dependency**
4. **Transitive functional dependency**

Trivial FD

- In **Trivial functional dependency**, a dependent is always **a subset** of the determinant.
- In other words, a functional dependency is called trivial if the attributes on the right side are the subset of the attributes on the left side of the functional dependency.
- **X → Y** is called a trivial functional dependency if **Y** is the subset of **X**.

Non-Trivial FD

- It is the opposite of Trivial functional dependency.
- Formally speaking, in **Non-Trivial FD**, dependent if **not a subset** of the determinant.
- **X → Y** is called a Non-trivial functional dependency if **Y** is **not a subset** of **X**.
- So, a functional dependency **X → Y** where **X** is a set of attributes and **Y** is also a set of the attribute but not a subset of **X**, then it is called *Non-trivial functional dependency*.

Multivalued FD

- In **Multivalued fd**, attributes in the dependent set are **not dependent** on each other.

Transitive FD

- Consider two functional dependencies **A → B** and **B → C** then according to the *transitivity axiom* **A → C** must also exist. This is called a transitive functional dependency.
- In other words, dependent is indirectly dependent on determinant in Transitive functional dependency.

Armstrong's Axioms/Properties of FD

- William Armstrong in *1974* suggested a few rules related to functional dependency.
- They are called **RAT** rules.

1. **Reflexivity**:

- If **A** is a set of attributes and **B** is a subset of **A**, then the functional dependency **A → B** holds true.

2. **Augmentation**:

- If a functional dependency **A → B** holds true, then appending any number of the attribute to both sides of dependency doesn't affect the dependency.
- It remains true.

3. **Transitivity**:

- If two functional dependencies **X → Y** and **Y → Z** hold true, then **X → Z** also holds true by the rule of Transitivity.

Advantages of FD

1. It is used to **maintain the quality of data in the database**.
2. It **expresses the facts about the database design**.
3. It **helps in clearly defining the meanings and constraints of databases.**
4. It **helps to identify bad designs.**
5. Functional Dependency **removes data redundancy** where the same values should not be repeated at multiple locations in the same database table.
6. The process of Normalization starts with identifying the candidate keys in the relation. **Without functional dependency, it's impossible to find candidate keys and normalize the database.**

Conclusion

- Functional dependency defines how the attributes of a relation are related to each other.
- It helps in maintaining the quality of data in the database.
- It is denoted by an arrow "→".
- The functional dependency of **A** on **B** is represented by A → B.
- **William Armstrong** in *1974* suggested a few axioms or rules related to functional dependency.
- They are
    - *Rule of Reflexivity*
    - *Rule of Augmentation*
    - *Rule of Transitivity*
- There are four types of functional dependency in DBMS - **Trivial**, **Non-Trivial**, **Multivalued** and **Transitive functional dependency**.
- Functional dependencies have many advantages, keeping the database design clean, defining the **meaning and constraints** of the databases, and removing **data redundancy** are a few of them.

Normalization

Overview

- Normalization is the process of organizing the data and the attributes of a database.
- It is performed to reduce the data redundancy in a database and to ensure that data is stored logically.
- Data redundancy in DBMS means having the same data but at multiple places.
- It is necessary to remove data redundancy because it causes anomalies in a database which makes it very hard for a database administrator to maintain it.

Why Do We Need Normalization?

- Normalization is used to reduce data redundancy.
- It provides a method to remove the following anomalies from the database and bring it to a more consistent state:

A database anomaly is a flaw in the database that occurs because of poor planning and redundancy.

1. **Insertion anomalies**: This occurs when we are not able to insert data into a database because some attributes may be missing at the time of insertion.
2. **Updation anomalies:** This occurs when the same data items are repeated with the same values and are not linked to each other.
3. **Deletion anomalies:** This occurs when deleting one part of the data deletes the other necessary information from the database.

Normal Forms

There are four types of normal forms that are usually used in relational databases as you can see:

1. **1NF:** A relation is in 1NF if all its attributes have an atomic value.
2. **2NF:** A relation is in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the candidate key in DBMS.
3. **3NF:** A relation is in 3NF if it is in 2NF and there is no transitive dependency.
4. **BCNF:** A relation is in BCNF if it is in 3NF and for every Functional Dependency, LHS is the super key.

To understand the above-mentioned normal forms, we first need to have an understanding of the functional dependencies.

- **Functional dependency** is a relationship that exists between two sets of attributes of a relational table where one set of attributes can determine the value of the other set of attributes.
- It is denoted by X -> Y, where X is called a determinant and Y is called dependent.

First Normal Form (1NF)

- A relation is in 1NF if every attribute is a single-valued attribute or it does not contain any multi-valued or composite attribute, i.e., every attribute is an atomic attribute.
- If there is a composite or multi-valued attribute, it violates the 1NF.
- To solve this, we can create a new row for each of the values of the multi-valued attribute to convert the table into the 1NF.

Second Normal Form (2NF)

- The normalization of 1NF relations to 2NF involves the elimination of partial dependencies.
- A partial dependency in DBMS exists when any non-prime attributes, i.e., an attribute not a part of the candidate key, is not fully functionally dependent on one of the candidate keys.

For a relational table to be in second normal form, it must satisfy the following rules:

1. The table must be in first normal form.
2. It must not contain any partial dependency, i.e., all non-prime attributes are fully functionally dependent on the primary key.

If a partial dependency exists, we can divide the table to remove the partially dependent attributes and move them to some other table where they fit in well.

The prime attributes in DBMS are those which are part of one or more candidate keys.

- The relations in 2NF are clearly less redundant than relations in 1NF.
- However, the decomposed relations may still suffer from one or more anomalies due to the transitive dependency.
- We will remove the transitive dependencies in the Third Normal Form.

Third Normal Form (3NF)

- The normalization of 2NF relations to 3NF involves the elimination of transitive dependencies in DBMS.

A functional dependency X -> Z is said to be transitive if the following three functional dependencies hold:

- X -> Y
- Y does not -> X
- Y -> Z

For a relational table to be in third normal form, it must satisfy the following rules:

1. The table must be in the second normal form.
2. No non-prime attribute is transitively dependent on the primary key.

3. For each functional dependency X -> Z at least one of the following conditions hold:

- X is a super key of the table.
- Z is a prime attribute of the table.

If a transitive dependency exists, we can divide the table to remove the transitively dependent attributes and place them to a new table along with a copy of the determinant.

- The 2NF and 3NF impose some extra conditions on dependencies on candidate keys and remove redundancy caused by that.
- However, there may still exist some dependencies that cause redundancy in the database.
- These redundancies are removed by a more strict normal form known as BCNF.

Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form(BCNF) is an advanced version of 3NF as it contains additional constraints compared to 3NF.

For a relational table to be in BCNF, it must satisfy the following rules:

1. The table must be in the third normal form.
2. For every non-trivial functional dependency X -> Y, X is the superkey of the table. That means X cannot be a non-prime attribute if Y is a prime attribute.

- A superkey is a set of one or more attributes that can uniquely identify a row in a database table.

Conclusion

- Normal forms are a mechanism to remove redundancy and optimize database storage.
- In 1NF, we check for atomicity of the attributes of a relation.
- In 2NF, we check for partial dependencies in a relation.
- In 3NF, we check for transitive dependencies in a relation.
- In BCNF, we check for the superkeys in LHS of all functional dependencies.

Codd's 12 Rules

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

1: **Information Rule**

- The data stored in a database, may it be user data or metadata, must be a value of some table cell.

- Everything in a database must be stored in a table format.

2: **Guaranteed Access Rule**

- Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).
- No other means, such as pointers, can be used to access data.

3: **Systematic Treatment of NULL Values**

- The NULL values in a database must be given a systematic and uniform treatment.
- This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

4: **Active Online Catalog**

- The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users.
- Users can use the same query language to access the catalog which they use to access the database itself.

5: **Comprehensive Data Sub-Language Rule**

- A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.
- This language can be used directly or by means of some application.
- If the database allows access to data without any help of this language, then it is considered as a violation.

6: **View Updating Rule**

- All the views of a database, which can theoretically be updated, must also be updatable by the system.

7: **High-Level Insert, Update, and Delete Rule**

- A database must support high-level insertion, updation, and deletion.
- This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

8: **Physical Data Independence**

- The data stored in a database must be independent of the applications that access the database.
- Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

9: **Logical Data Independence**

- The logical data in a database must be independent of its user's view (application).
- Any change in logical data must not affect the applications using it.
- This is one of the most difficult rule to apply.

For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application.

10: **Integrity Independence**

- A database must be independent of the application that uses it.
- All its integrity constraints can be independently modified without the need of any change in the application.
- This rule makes a database independent of the front-end application and its interface.

11: **Distribution Independence**

- The end-user must not be able to see that the data is distributed over various locations.
- Users should always get the impression that the data is located at one site only.
- This rule has been regarded as the foundation of distributed database systems.

12: **Non-Subversion Rule**

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.