Functional Dependency

Functional dependency \( FD \) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes \( A_1, A_2, ..., A_n \), then those two tuples must have to have same values for attributes \( B_1, B_2, ..., B_n \).

Functional dependency is represented by an arrow sign \( \rightarrow \) that is, \( X \rightarrow Y \), where \( X \) functionally determines \( Y \). The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

If \( F \) is a set of functional dependencies then the closure of \( F \), denoted as \( F^+ \), is the set of all functional dependencies logically implied by \( F \). Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** – If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.

- **Augmentation rule** – If \( a \rightarrow b \) holds and \( y \) is attribute set, then \( ay \rightarrow by \) also holds. That is adding attributes in dependencies, does not change the basic dependencies.

- **Transitivity rule** – Same as transitive rule in algebra, if \( a \rightarrow b \) holds and \( b \rightarrow c \) holds, then \( a \rightarrow c \) also holds. \( a \rightarrow b \) is called as a functionally that determines \( b \).

Trivial Functional Dependency

- **Trivial** – If a functional dependency \( FD \ X \rightarrow Y \) holds, where \( Y \) is a subset of \( X \), then it is called a trivial FD. Trivial FDs always hold.

- **Non-trivial** – If an \( FD \ X \rightarrow Y \) holds, where \( Y \) is not a subset of \( X \), then it is called a non-trivial FD.

- **Completely non-trivial** – If an \( FD \ X \rightarrow Y \) holds, where \( x \) intersect \( Y = \emptyset \), it is said to be a completely non-trivial FD.

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

**First Normal Form**

First Normal Form is defined in the definition of relations **tables** itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.
We re-arrange the relation table as below, to convert it to First Normal Form.

<table>
<thead>
<tr>
<th>Course</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming</td>
<td>Java, c++</td>
</tr>
<tr>
<td>Web</td>
<td>HTML, PHP, ASP</td>
</tr>
</tbody>
</table>

We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.
We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

**Third Normal Form**

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy –

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, \( X \rightarrow A \), then either –
  - \( X \) is a superkey or,
  - \( A \) is prime attribute.

**Student_Detail**

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows –

**Boyce-Codd Normal Form**

Boyce-Codd Normal Form **BCNF** is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency, \( X \rightarrow A \), \( X \) must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

\[
\text{Stu_ID} \rightarrow \text{Stu_Name}, \text{Zip}
\]

and

\[
\text{Zip} \rightarrow \text{City}
\]

Which confirms that both the relations are in BCNF.