NORMAL FORM

Normalization is the set of guidelines used to optimally design a database to reduce redundant data. Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

A database that is not normalized may include data that is contained in one or more different tables for no apparent reasons. Normalization is achieved by applying a number of tests called normal forms to tables.

Goals of normalization

The goals of normalization are:

1. Eliminating data redundancy i.e. duplication of data
2. To minimize or avoid modification issues
3. To simplify queries

Normal form

Normal form is the way of measuring the level, or depth, to which a database has been normalized. The most common normal forms are:

1. First normal form (1NF)
2. Second normal form (2NF)
3. Third normal form (3NF)

Other normal forms are Boyce /Codd normal form (BCNF), 4^{th} normal form and 5^{th} normal form.

Note: Each subsequent normal form depends on the normalization steps taken in the previous normal form. The 3NF is sufficient for most typical database applications.

First normal form (1NF)

A database table is said to be in 1NF if:

- It contains no repeating or duplicate fields/columns (i.e. each column name should be unique)
- No data in a columns is multi-valued (i.e. each data field is single value)
- Each row of data has a unique identifier (or Primary Key)
- The attribute domain remains the same (i.e. age value cannot be in the name column)

For example, consider the un-normalized table below**:**

Item	**Colour**	**Price**	**Tax**
T-shirt	Red, Blue	12.00	0.60
Polo	Red, Yellow	12.00	0.60
T-shirt	Red, Blue	12.00	0.60
Sweatshirt	Blue, Black	25.00	1.25
Pant	White	6.00	0.30

Table1: **Un-normalized table**

The table above is not in a first normal form because:

- Multiple items in color field (i.e. red and blue , red and yellow etc)
- Duplicate records/no particular primary key (i.e. row1 and row3)

So, how do you convert the table above into 1NF?

- Delete one of the duplicate records (i.e. row1 or row3)
- Expand the remaining rows such that each column has a single value

The resulting table now in 1NF is shown below.

Item	Colour	Price	Tax
T-shirt	Red	12.00	0.60
T-shirt	Blue	12.00	0.60
Polo	Red	12.00	0.60
Polo	Yellow	12.00	0.60
Sweatshirt	Blue	25.00	1.25
Sweatshirt	Black	25.00	1.25
Pant	White	6.00	0.30

Table 2: **1NF table**

Problems with tables in first normal form (1NF)

Insert anomalies: An **Insert Anomaly** occurs when certain attributes cannot be inserted into the database without the presence of other attributes. Suppose a new **item** has just been bought and is to be added into the table and you do not know the **tax** applicable, it will be difficult to enter a few item of information and not all, thereby leading to Insertion Anomaly.

Delete anomalies: A **Delete Anomaly** exists when certain attributes are lost because of the deletion of other attributes. If the **White** colour of the item **Pant** is no more in stock**,** and we try to delete '**white'** from the colour column, then we will be forced to remove the item Pant, the price and the tax as well since the entire row will be deleted

Update anomalies: An **Update Anomaly** exists when one or more instances of duplicated data are updated, but not all. For example, if the **tax** applicable to the price **12.0** changed, then we will have to update all the rows where there is **0.60**, else data will become inconsistent i.e. there will be different **tax** value for the price **12.0** in different rows.

Second normal form (2NF)

A database table is in a 2NF if and only if:

1. It is in 1NF and
2. Every non-key attribute is fully functionally dependent on the primary key i.e. there should be no partial dependency in the table

Consider the Table 2 above, the non-key attributes are **colour**, **price** and **tax**. The attribute item is the primary key. The price and the tax are functionally dependent on the item and not on the colour i.e. the item determines the price and not the colour.

So what do you do to normalize into 2NF?
Decompose the **1NF** table and set up a new relation (table) for each partial key with its dependent attributes. Make sure to keep a table with the original primary key and any attributes that are fully functionally dependent on it.

Having decomposed the above table, the tables below are now in **2NF**

2NF(a)

Item	Colour
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Sweatshirt	Blue
Sweatshirt	Black
Pant	White

Item	Price	Tax
T-shirt	12.00	0.60
Polo	12.00	0.60
Sweatshirt	25.00	1.25
Pant	6.00	0.30

2NF(b)

Tables 3: 2NF Tables

Third normal form (3NF)

The third normal form's objective is to remove data in a table that is not dependent on the primary key.
A database table is said to be in 3NF if:

- It is in a 2NF
- All non-key field depend only on the primary key – no transitive dependency (i.e. Eliminate all fields that do not depend on the primary key by moving them into a separate table)

Tables 2NF(b) is not in third normal form because **Tax** depends on **price**, not **item.**

To normalize to 3NF, decompose the table and set up a new table that includes the non-key attribute(s) i.e. **price,** that functionally determine(s) other non-key attribute(s) i.e. **tax**

The tables below are now in 3NF.

2NF(a)

Item	Colour
T-shirt	Red
T-shirt	Blue
Polo	Red
Polo	Yellow
Sweatshirt	Blue
Sweatshirt	Black
Pant	White

Item	Price	Tax
T-shirt	12.00	0.60
Polo	12.00	0.60
Sweatshirt	25.00	1.25
Pant	6.00	0.30

2NF(b)

Price	Tax
12.00	0.60
25.00	1.25
6.00	0.30

3NF