

1. Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process –

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

I. A first normal form

The First normal form (1NF) sets basic rules for an organized database –

- Define the data items required, because they become the columns in a table.
- Place the related data items in a table.
- Ensure that there are no repeating groups of data.
- Ensure that there is a primary key.

First Rule of 1NF

You must define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains and then finally putting the related columns into their own table.

For example, you put all the columns relating to locations of meetings in the Location table, those relating to members in the MemberDetails table and so on.

Second Rule of 1NF

The next step is ensuring that there are no repeating groups of data. Consider we have the following table –

```
CREATE TABLE CUSTOMERS(
  ID INT NOT NULL,
  NAME VARCHAR (20) NOT NULL,
  AGE INT NOT NULL,
  ADDRESS CHAR (25),
  ORDERS VARCHAR(155)
);
```

So, if we populate this table for a single customer having multiple orders, then it would be something as shown below –

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200
100	Sachin	36	Lower West Side	Tripod Large

But as per the 1NF, we need to ensure that there are no repeating groups of data. So, let us break the above table into two parts and then join them using a key as shown in the following program –

CUSTOMERS table –

```
CREATE TABLE CUSTOMERS(
  ID INT NOT NULL,
```

```
NAME VARCHAR (20) NOT NULL,  
AGE INT NOT NULL,  
ADDRESS CHAR (25),  
PRIMARY KEY (ID)  
);
```

This table would have the following record –

ID	NAME	AGE	ADDRESS
100	Sachin	36	Lower West Side

ORDERS table –

```
CREATE TABLE ORDERS(  
ID INT NOT NULL,  
CUSTOMER_ID INT NOT NULL,  
ORDERS VARCHAR(155),  
PRIMARY KEY (ID)  
);
```

This table would have the following records –

ID	CUSTOMER_ID	ORDERS
10	100	Cannon XL-200
11	100	Battery XL-200
12	100	Tripod Large

Third Rule of 1NF

The final rule of the first normal form, create a primary key for each table which we have already created.

II. A Second normal form

The Second Normal Form states that it should meet all the rules for 1NF and there must be no partial dependences of any of the columns on the primary key –

Consider a customer-order relation and you want to store customer ID, customer name, order ID and order detail and the date of purchase –

```
CREATE TABLE CUSTOMERS(  
  CUST_ID INT NOT NULL,  
  CUST_NAME VARCHAR (20) NOT NULL,  
  ORDER_ID INT NOT NULL,  
  ORDER_DETAIL VARCHAR (20) NOT NULL,  
  SALE_DATE DATETIME,  
  PRIMARY KEY (CUST_ID, ORDER_ID)  
);
```

This table is in the first normal form; in that it obeys all the rules of the first normal form. In this table, the primary key consists of the CUST_ID and the ORDER_ID. Combined, they are unique assuming the same customer would hardly order the same thing.

However, the table is not in the second normal form because there are partial dependencies of primary keys and columns. CUST_NAME is dependent on CUST_ID and there's no real link between a customer's name and what he purchased. The order detail and purchase date are also dependent on the ORDER_ID, but they are not dependent on the CUST_ID, because there is no link between a CUST_ID and an ORDER_DETAIL or their SALE_DATE.

To make this table comply with the second normal form, you need to separate the columns into three tables.

First, create a table to store the customer details as shown in the code block below –

```
CREATE TABLE CUSTOMERS(  
  CUST_ID INT NOT NULL,  
  CUST_NAME VARCHAR (20) NOT NULL,  
  PRIMARY KEY (CUST_ID)  
);
```

The next step is to create a table to store the details of each order –

```
CREATE TABLE ORDERS(  
  ORDER_ID INT NOT NULL,  
  ORDER_DETAIL VARCHAR (20) NOT NULL,  
  PRIMARY KEY (ORDER_ID)  
);
```

Finally, create a third table storing just the CUST_ID and the ORDER_ID to keep a track of all the orders for a customer –

```
CREATE TABLE CUSTMERORDERS(  
  CUST_ID INT NOT NULL,  
  ORDER_ID INT NOT NULL,  
  SALE_DATE DATETIME,
```

```
PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

III. A third normal form

A table is in a third normal form when the following conditions are met –

- It is in second normal form.
- All nonprimary fields are dependent on the primary key.

The dependency of these non-primary fields is between the data. For example, in the following table – the street name, city and the state are unbreakably bound to their zip code.

```
CREATE TABLE CUSTOMERS(
  CUST_ID      INT          NOT NULL,
  CUST_NAME    VARCHAR(20)  NOT NULL,
  DOB          DATE,
  STREET       VARCHAR(200),
  CITY         VARCHAR(100),
  STATE        VARCHAR(100),
  ZIP          VARCHAR(12),
  EMAIL_ID     VARCHAR(256),
  PRIMARY KEY (CUST_ID)
);
```

The dependency between the zip code and the address is called as a transitive dependency. To comply with the third normal form, all you need to do is to move the Street, City and the State fields into their own table, which you can call as the Zip Code table. –

```
CREATE TABLE ADDRESS(
  ZIP          VARCHAR(12),
  STREET       VARCHAR(200),
  CITY         VARCHAR(100),
  STATE        VARCHAR(100),
  PRIMARY KEY (ZIP)
);
```

The next step is to alter the CUSTOMERS table as shown below –

```
CREATE TABLE CUSTOMERS(
```

```
CUST_ID    INT          NOT NULL,  
CUST_NAME  VARCHAR(20)  NOT NULL,  
DOB        DATE,  
ZIP        VARCHAR(12),  
EMAIL_ID   VARCHAR(256),  
PRIMARY KEY (CUST_ID)  
);
```

The advantages of removing transitive dependencies are mainly two-fold. First, the amount of data duplication is reduced and therefore your database becomes smaller.

The second advantage is data integrity. When duplicated data changes, there is a big risk of updating only some of the data, especially if it is spread out in many different places in the database.

For example, if the address and the zip code data were stored in three or four different tables, then any changes in the zip codes would need to ripple out to every record in those three or four tables.