# Normalization

# What is Normalization?

# Normalization is a process in database design aimed at organizing data to reduce redundancy and improve data integrity. It involves breaking down a database into multiple related tables and defining relationships between them to ensure that each table focuses on a single subject or theme.

# Goals of Normalization

# Minimize Redundancy: Prevents duplication of data.

# Ensure Data Integrity: Maintains consistency and correctness of data.

# Enhance Scalability: Makes it easier to adapt the database to future changes.

# Facilitate Maintenance: Simplifies updates and reduces the likelihood of errors.

# Normalization Process for Food Products Database

This document outlines the step-by-step normalization process applied to the food products database. Normalization ensures the database adheres to data integrity rules and eliminates redundancy by dividing the database into multiple related tables.

## 1NF: First Normal Form

1NF (First Normal Form) is a fundamental concept in database normalization, which aims to reduce redundancy and improve data integrity. A database is said to be in 1NF if it satisfies the following conditions:

1. **Atomicity of Values**: Each column must contain atomic (indivisible) values. This means that no column should have multiple values, arrays, or lists.
2.  **Unique Rows**: Each row in a table must be unique and identifiable by a primary key.
3.  **Consistent Column Data Types**: All entries in a column must be of the same type.

SQL Example for 1NF:

create table food\_products (

Product\_Id varchar(10),

Product\_Name varchar(100),

Category varchar(50),

Price decimal(10, 2),

Stock\_Quantity int,

Supplier varchar(100),

Expiration\_Date date,

Discount\_Percentage int,

Is\_Organic varchar(3),

Rating decimal(2, 1)

);

The above table is not in a 1NF form because it doesn’t consists of any primary key, so we need to add a primary key to the table so the data satisfies the 1NF form:

alter table food\_products

add primary key (Product\_Id);

after executing the above query now our data satisfies the 1st Normal Form.

## 2NF: Second Normal Form

In **2NF**, we break the table into multiple smaller tables to ensure no non-key attribute depends only on part of the primary key. The primary key in this case is product\_id.

We will split the food\_products table into:

1. A **products** table with product-specific information.
2. A **suppliers** table with supplier-specific details.

SQL Example for 2NF:

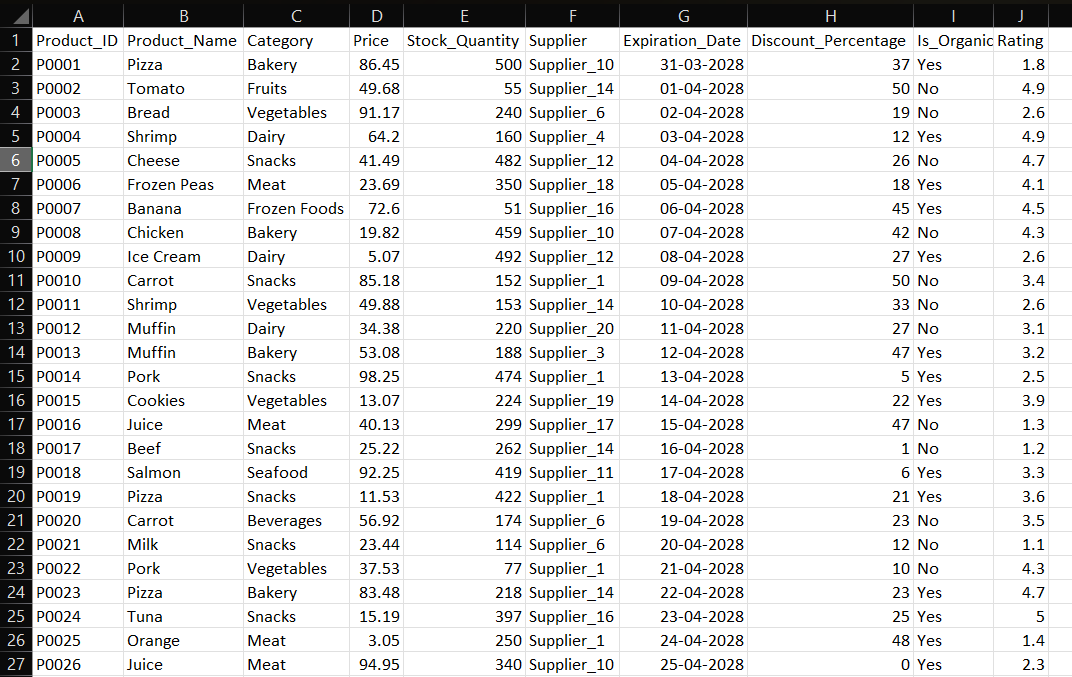
**Create a Table for Unique Products**

This table contains product-related details, removing supplier-related and category-related data.

create table products as

select product\_id,product\_name,category,price,stock\_quantity,supplier,expiration\_date,discount\_percentage,is\_organic,rating

from food\_products;



**Create a Suppliers Table**

This table contains unique supplier-related details. Each supplier will have a unique supplier\_id.

create table suppliers

as select distinct supplier from food\_products;

If needed, add an auto-incremented supplier\_id to make it the primary key.

alter table suppliers

add supplier\_id serial primary key;

Now alter the table product

alter table products add supplier\_id int;

**Link Products to Suppliers**

Now, update the products table to reference the supplier\_id from the suppliers table.

update products as p

set supplier\_id = s.supplier\_id

from suppliers as s

where p.supplier = s.supplier;

Finally, drop the supplier column from the products table to avoid redundancy:

alter table products drop column supplier;

**Join Products and Suppliers**

select p.\*, s.supplier

from products as p

join suppliers as s

on p.supplier\_id = s.supplier\_id;

## 3NF: Third Normal Form

**Identifying Transitive Dependencies**

1. In the products table:
   * category might have repeating values and could depend on the product itself.
   * To remove transitive dependencies, we can normalize category into its own table.

### **Split** products **into Separate Tables**

#### Create a categories Table:

A separate table for product categories.

create table categories

as select distinct category from food\_products;

**Insert Unique Categories:**

Insert distinct categories from the products table.

alter table categories

add category\_id serial primary key;

**Add category\_id to the products Table:**

Link the products table to the categories table via category\_id.

1. Add the category\_id column:

alter table products add category\_id int;

1. Populate category\_id based on categories:

update products as p

set category\_id = c.category\_id

from categories as c

where p.category = c.category;

1. Drop the redundant category column:

alter table products drop column category;

**Reconstructing Original Data**

To join all tables and get the original structure:

select p.\*,s.supplier,c.category

from products p

join categories c on p.category\_id = c.category\_id

join suppliers s on p.supplier\_id = s.supplier\_id;

Now finally add the foreign keys to maintain the relationship between data:

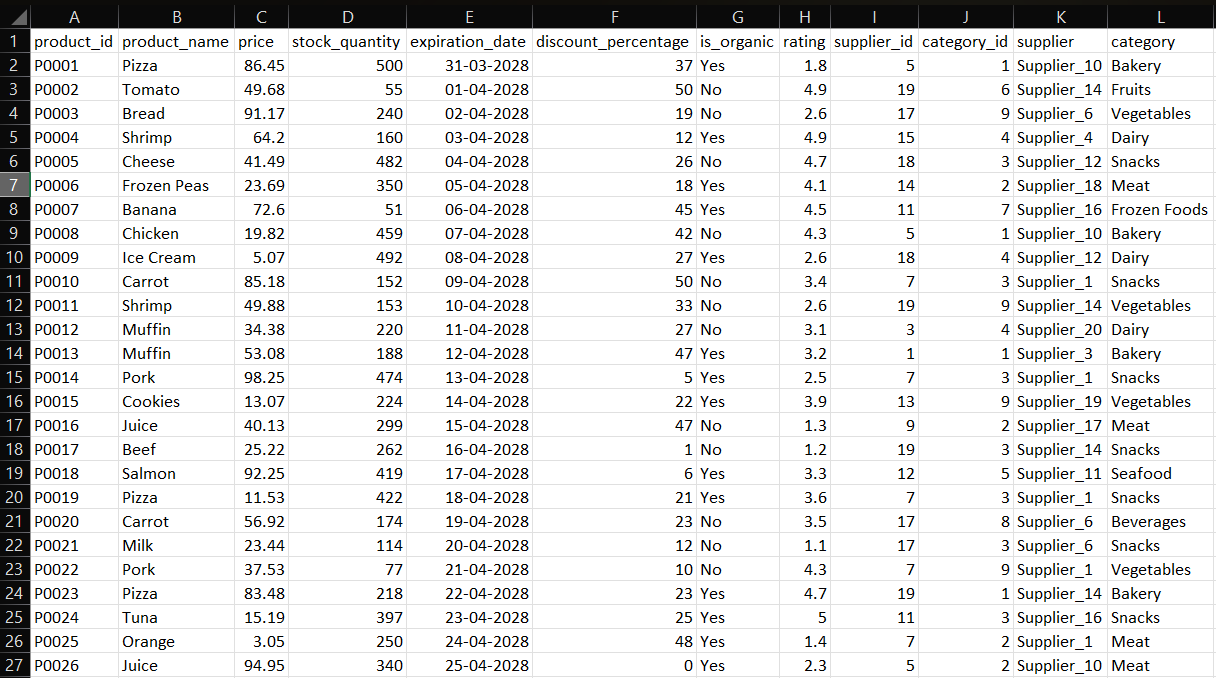
alter table products

add foreign key (category\_id)references categories (category\_id);

alter table products

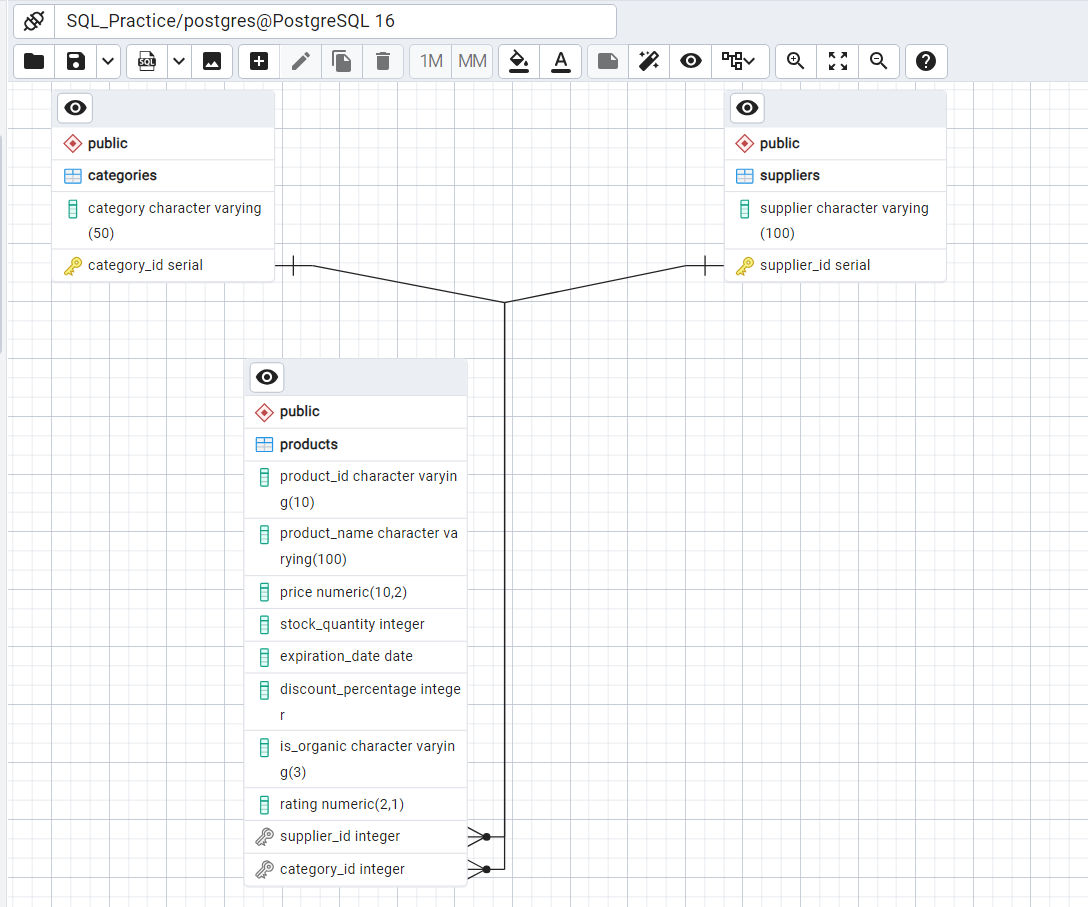
add foreign key (supplier\_id)references suppliers (supplier\_id);

Final output after doing Normalization:



## ERD(Entity-Relationship Diagram):

ERD for the given data is as following:



## Conclusion

By normalizing the food products database to 3NF, we have ensured minimal redundancy, improved data integrity, and established clear relationships between the tables. The resulting schema is efficient and adheres to best database design practices.