Database Normalization: Beyond 3NF

In my article on Database Normalization, I walked through the process of transforming a 0NF database design through to 3NF, at each step examining query performance as well as improvements in data integrity safeguards. This document picks up where that article left off, and explains the process of transforming the design through 4NF and 5NF.

Over the course of refactoring a database to 3NF, we will satisfy the requirements of most applications in terms of query performance and preservation of data quality.

So why do the other, higher Normal Forms exist? The short answer is that they deal with certain "permutations of attributes". In our example, this means permutations of the Product, Customer and Size attributes. Not all permutations may exist; for example, some products (such as an umbrella) may have a size, but no gender-specific target customer group. Another product, such as lipstick, may have a target customer but no size.

With the 3NF design we are forced to store the non-existing counterpart for the Customer-Size permutations, and deal with the fact that some ID (in the Sizes table and Customers table) means "no value".

Fourth Normal Form – Isolate Independent Multiple Relationships

In 4NF, no table may contain two or more 1-to-many, or many-to-many, relationships that are not directly related. Our current design flouts this rule because one product may come in multiple sizes, and many products may use many sizes. A common mistake here is to use a design like that shown in Figure 7. The objective is to have the all permutations of customers and sizes that are available for a certain product in the table.
Figure 7: An incorrect 4NF design

It can work, if all products have a customer group and a size. In most cases, this is not true and so this design will break 4NF. In order to implement the design from Figure 7 in our example, the `ProductsCustomersSizes` table would look as shown in Figure 8. **This is also exactly how Article table is designed at this stage.**
For the missing attributes, we cannot store NULL values (with all the associated problems with three-tier logic in terms of writing queries), so we're forced to use "dummy" values, which could mean either "n/a", "all" or "none" to avoid that particular problem. Whatever you choose, make sure everyone in the project knows the meaning of the "dummy" value. Figure 9 shows a much better design, for 4NF compliance.

Figure 9: A correct 4NF design

Script 3NF-4NF.sql, in the code download, provides the scheme changes needed to convert our 3NF design into the 4NF. Listing 8 shows the versions of the three queries, for the 4NF design.

```sql
/*QUERY 1: Number of Unique Products*/
SELECT  
p.Product ,
       m.Manufacturer ,
       c.Color
FROM     dbo.Products AS p
         INNER JOIN dbo.Manufacturers AS m ON m.ManufacturerID = p.ManufacturerID
         INNER JOIN dbo.Colors AS c ON c.ColorID = p.ColorID

/*QUERY 2: Number of pink products, targeted for Males in Size 122*/
SELECT  
p.Product ,
       m.Manufacturer ,
       c.Color
FROM     dbo.Products AS p
```
INNER JOIN dbo.Manufacturers AS m ON m.ManufacturerID = p.ManufacturerID
INNER JOIN dbo.Colors AS c ON c.ColorID = p.ColorID
AND c.Color = 'Pink'
INNER JOIN dbo.Customers AS x ON x.CustomerID = pc.CustomerID
AND x.Customer = 'Male'
INNER JOIN dbo.Sizes AS s ON s.SizeID = ps.SizeID
AND s.Size = '122'

/*QUERY 3: Get all unique sizes*/
SELECT Size
FROM dbo.Sizes
WHERE Size > ''

Listing 8: 4NF queries

Let's see how this redesign for 4NF affected the performance of our queries.

<table>
<thead>
<tr>
<th>Query</th>
<th>CPU (ms)</th>
<th>Duration (ms)</th>
<th>Reads</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4NF</td>
<td>3NF</td>
<td>4NF</td>
<td>3NF</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>189</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4NF</td>
<td>3NF</td>
<td>4NF</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>15</td>
<td>109</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 9: 4NF query performance, compared to 3NF

The single largest schema change in this 4NF design is that the Articles table is no longer used! The permutations of the required attributes has been split out into tables of their own, as shown in Figure 10.
Figure 10: The `ProductsCustomers` and `ProductsSizes` tables in the 4NF design

With this design, we can easily see which customer group is the target for which product, and which size is related to which product – and we do not need to use "n/a" values nor NULL. We only store actual value and thus comply with the fourth rule from Date. Now it is also easy to see if some products have a customer but not a size.

This design makes it obvious that `Customer` and `Size` are independently linked to `Products`. It also means we miss information about which sized are available to which customers. In 4NF we can only tell if Male and/or Female is associated to a certain product and also which sizes are available for the same product. We can not tell which sized belong to which customers. So how do we design to comply with this business rule? You guessed it right, we have to move to 5NF.

**Fifth Normal Form – Isolate Semantically Related Multiple Relationships**

In a 5NF design, we deal with any practical constraints on information that justify separating logically related many-to-many relationships. As we go further and further down the line for Normal Forms, it usually means that we need more tables to store our data in order for the data to be consistent and safe for deletes, updated and inserts.
Now consider this initial business request: we also want to know which sizes are available for which customers.

![Diagram of Customers, Products, and Sizes](image1)

**Figure 11: An incorrect 4NF design**

Storing the data as we first tried in the Fourth Normal Form (figure 7) is only viable when all products share the same customers and sizes (i.e. all permutations between them). The way to store the data without breaking any Normal Form is as shown in Figure 12.

![Diagram of ProductsCustomers, Products, ProductsSizes, Customers, CustomersSizes, Sizes](image2)

**Figure 12: The 5NF Design**

With this design, we can have the union of each customer and size for any Product (if present) or only the one or other (customer or size) for the product.
Figure 12: New tables in the 5NF Design

Script d 4NF-5NF.sql, in the code download, provides the scheme changes needed to convert our 4NF design into 5NF. Listing 8 shows how our three queries are designed for 5NF.

```sql
/*QUERY 1: Number of Unique Products*/
SELECT p.Product , m.Manufacturer , c.Color
FROM dbo.Products AS p
INNER JOIN dbo.Manufacturers AS m
    ON m.ManufacturerID = p.ManufacturerID
INNER JOIN dbo.Colors AS c ON c.ColorID = p.ColorID

/*QUERY 2: Number of pink products, targeted for Males in Size 122*/
SELECT p.Product , m.Manufacturer , c.Color
FROM dbo.Products AS p
INNER JOIN dbo.Manufacturers AS m
    ON m.ManufacturerID = p.ManufacturerID
INNER JOIN dbo.Colors AS c ON c.ColorID = p.ColorID
    AND c.Color = 'Pink'
INNER JOIN dbo.ProductsCustomers AS pc
    ON pc.ProductID = p.ProductID
```
**Listing 8: 5NF queries**

Let's see how this redesign for 5NF affected the performance of our queries.

<table>
<thead>
<tr>
<th>Query</th>
<th>CPU (ms)</th>
<th>Duration (ms)</th>
<th>Reads</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>68</td>
<td>18144</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>16</td>
<td>8843</td>
<td>1464</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>37</td>
</tr>
</tbody>
</table>

Figure 9: 5NF query performance, compared to 4NF

The performance figures are similar to those for the 3NF design, confirming that we can gain no additional performance benefits from further normalization, in this case.

**Sixth Normal Form**

Sixth Normal Form is intended for use with temporal data or intervals. For example, if a table contains an order's, we may also want to add temporal data, such as the time during which these values are, or were, valid (e.g., for historical data).

The values may vary independently of each other and at different rates. For example, we may wish to trace the history of changes to Status.