Applying the Normalization Process

We start by placing the fields of our sample pizza sales order (not pictured here) into a relation, which means we represent it as if it is a two-dimensional table. As you work through the normalization process, you will be rewriting existing relations and creating new ones. Some find it useful to draw the relations with sample tuples (rows) of data in them to assist in visualizing the work.

Invoice Number	Order Date	Customer Number	Customer Last Name	Customer First name	Street Address	City	State	Zip	Phone	Quantity	Notes	Pizza ID	Pizza Description	Pizza Size	Unit Price	Extended Amount	Order Total
56982	5/15/2015	12358	Lee	Nicole	1234 Mulberry St	Boise	ID	16845	232-448-0189	2	Be sure to deliver with extra napkins please	CHEESE1	Cheese Pizza	Medium	7.25	14.50	14.50
56983	5/15/2015	25846	Walker	Samuel	5 Brooks Place	Spokane	WA	88962	801-568-4151	1		PEP1	Pepperoni Pizza	Large	11.50	11.50	11.50

Table 1. Invoice in tabular form

Rewriting tables with sample data is a time-consuming process, so for the sake of saving some time, we'll simply write the attributes as a list and picture them in our minds as two-dimensional tables. Here is the list of attributes for our sample invoice data in Table 1 above:

INVOICE: Invoice Number, Order Date, Customer Number, Customer Last Name, Customer First Name, Street Address, City, State, Zip, Phone, Quantity, Notes, Pizza ID, Pizza Description, Pizza Size, Unit Price, [1]Extended Amount, [1]Order Total

Normalizing Your Database: First Normal Form (1NF)

Definition: A relation is said to be in First Normal Form (1NF) if and only if each attribute of the relation is atomic. More simply, to be in 1NF, the intersection of columns and rows in the relation can contain no more than one data value.

With our unnormalized relation complete we are now ready to get started with the normalization process. First normal form is probably the most important step in the entire normalization process as it facilitates the breaking up of our data into its related data groups, with the subsequent normal forms fine-tuning the relationships between and within the grouped data ("Relational Database Management Systems", n.d.)..

One of the goals of first normal form is to remove repeating groups. A repeating group is a domain or set of domains, directly relating to the key, that repeat data across tuples in order to cater for other domains where the data is different for each tuple such as with the line items on the invoice (*Pizza ID*, *Pizza Description*, *Pizza Size* and *Unit Price* and *Quantity* fields*)* ("Relational Database Management Systems", n.d.). By convention, we enclose repeating groups and multivalued attributes in pairs of parentheses. Rewriting our invoice in this way to show the line item data as a repeating group, we get this:

INVOICE: Invoice Number, Order Date, Customer Number, Customer Last Name, Customer First Name, Street Address, City, State, Zip, Phone, Notes, (Pizza ID, Pizza Description, Pizza Size, Unit Price, Quantity), [1]Extended Amount, [1]Order Total

It is essential to understand that although we know that our business has many customers, there is only one customer for any given invoice, so the customer data on the invoice is not a repeating group. You may have noticed that the customer data for a given customer is repeated on every invoice for that customer, but this is a problem that we will address when we get to third normal form. Because there is only one customer per invoice, the problem is not addressed when we transform the relation to first normal form.

[1]DO NOT INCLUDE the calculated attributes/fields (*Line Total*, *Subtotal*, *Sales Tax* and *Total*) in your normalization diagrams or queries.

To transform unnormalized relations into first normal form, we can create a relation that uses a composite key, then enter new invoice data for each customer into a new row. For our invoice example the *Invoice Number* and *Pizza ID* attributes will become the composite primary key. The table below shows this approach applied to the INVOICE relation:

```
INVOICE:          Invoice Number (PK), Customer Number, Customer Last
                  Name, Customer First Name, Street Address, City,
                  State, Zip, Phone, Notes, Order Date, Pizza ID (PK),
                  Pizza Description, Pizza Size, Unit Price, Quantity,
                  [1]Extended Amount, [1]Order Total
```

Normalizing Your Database: Second Normal Form (2NF)

Definition: A relation is said to be in second normal form if it meets both the following criteria:

- **The relation is in first normal form.**
- **All non-key attributes are functionally dependent on the entire primary key.**

Before we explore second normal form, let's review the concept of functional dependence. For this definition, we'll use two arbitrary attributes, cleverly named "A" and "B." First, if we say that attribute B is functionally dependent on attribute A, what we are also saying is that attribute A determines attribute B, or that A is a determinant (unique identifier) of attribute B. Second, let's look again at the first normal form relation in our invoice example:

In the INVOICE relation, we can see that *Customer Number*, *Customer Last Name*, *Customer First Name*, *Street Address*, *City*, *State*, *Zip*, *Phone*, *Notes*, *Order Date* and *Order Total* depend only on the *Invoice Number* instead of the combination of *Invoice Number* and *Pizza ID.* These fields are said to be functionally dependent on *Invoice Number* because at any point in time, there can be only one value associated with a given *Invoice Number;* however, because there is no dependency on *Pizza ID*, which is a part of the primary key, these fields are in violation of second normal form.

If we look at the *Pizza Description* or *Pizza Size* fields it should be easy to see that *Pizza ID* alone determines the value. Said another way, if the same pizza appears as a line item on many different invoices, the *Pizza Description* or *Pizza Size* is the same regardless of the *Invoice Number*. We can say that *Pizza Description* or *Pizza Size* is functionally dependent on only ***part of*** the primary key because it depends only on *Pizza ID* and not on the ***combination*** of *Invoice Number* ***and*** *Pizza ID*. This partial dependence is the very issue addressed by second normal form.

The only fields in the 1NF INVOICE relation that depend on the combination of *Invoice Number* and *Pizza ID* are the *Quantity* and *Extended Amount* fields.

Looking at the INVOICE relation the second normal form violations should be readily apparent:

1) *Pizza Description*, *Unit Price* and *Pizza Size* depend only on the *Pizza ID* instead of the combination of *Invoice Number* and *Pizza ID*; and
2) *Customer Number*, *Customer Last Name*, *Customer First Name*, *Street Address*, *City*, *State*, *Zip*, *Phone*, *Notes*, *Order Date* and *Order Total* depend only on the *Invoice Number* instead of the combination of *Invoice Number* and *Pizza ID.*

Now that we have identified the second normal form violations, the solution is to place the attributes that are partially dependent into separate relations where they depend on the entire key instead of part of the key. Here is our invoice example rewritten into second normal form:

```
INVOICE:          Invoice Number(PK), Customer Number, Customer Last
                  Name, Customer First Name, Street Address, City,
                  State, Zip, Phone, Notes, Order Date, [1]Order Total
```

[1]DO NOT INCLUDE the calculated attributes/fields (*Line Total*, *Subtotal*, *Sales Tax* and *Total*) in your normalization diagrams or queries.

INVOICE LINE ITEM:	Invoice Number(PK)(FK), Pizza ID(PK)(FK), Quantity, [1]Extended Amount
PIZZA:	Pizza ID (PK), Pizza Description, Pizza Size, Unit Price

Normalizing Your Database: Third Normal Form (3NF)

Definition: A relation is said to be in third normal form if it meets both the following criteria:

- **The relation is in second normal form.**
- **There is no transitive dependence (that is, all the non-key attributes depend only on the primary key).**

To understand third normal form, you must first understand transitive dependency. An attribute that depends on another attribute that is not the primary key of the relation is said to be transitively dependent. Looking at our INVOICE relation in second normal form, one can clearly see that *Customer Last Name* is dependent on *Invoice Number* (each *Invoice Number* has only one *Customer Last Name* value associated with it), but at the same time, *Customer Last Name* is also dependent on *Customer Number*. The same can be said of the rest of the customer attributes as well. The problem here is that attributes of the Customer entity have been included in our INVOICE relation.

To transform a second normal form relation into third normal form, simply move any transitively dependent attributes to a relation where they depend only on the primary key. Be careful to leave the attribute on which they depend in the original relation as a foreign key. You will need it to reconstruct the original user view via a join.

If you have been wondering about easily calculated attributes such as *Extended Amount* in the INVOICE LINE ITEM relation, it is actually third normal form that forbids them, but it takes a subtle interpretation of the rule. Because the *Extended Amount* is calculated by multiplying *Unit Price* by *Quantity*, it follows that *Extended Amount* is determined by the combination of *Unit Price* and *Quantity* and therefore is transitively dependent on those two attributes. Thus, it is third normal form that tells us to remove easily calculated attributes. [1]

Here is the invoice data rewritten into third normal form:

INVOICE:	Invoice Number (PK), Notes, Order Date, Customer Number (FK)
INVOICE LINE ITEM:	Invoice Number(PK)(FK), Pizza ID(PK)(FK), Quantity
PIZZA:	Pizza ID (PK), Pizza Description, Pizza Size, Unit Price
CUSTOMER:	Customer Number (PK), Customer Last Name, Customer First Name, Street Address, City, State, Zip, Phone

An easy way to remember the rules of first, second, and third normal form: In a third normal form relation, every non-key attribute must depend on the key, the whole key, and nothing but the key, so help me Codd. ☺

Sources:

Oppel, A. (2004). Logical Database Design Using Normalization. In *Databases demystified* (p. 145-160). New York: McGraw-Hill/Osborne.

Relational Database Management Systems. (n.d.) Retrieved from http://rdbms.opengrass.net/index.html.

[1]DO NOT INCLUDE the calculated attributes/fields (*Line Total*, *Subtotal*, *Sales Tax* and *Total*) in your normalization diagrams or queries.