

# The 3 Normal Forms: A Tutorial

by Fred Coulson

Copyright © Fred Coulson 2002, 2006

This tutorial may be freely copied and distributed, providing appropriate attribution to the author is given.

Inquiries may be directed to <http://phlonx.com/contact>

<http://phlonx.com/resources/nf3/>

## Table of Contents

<b>TABLE OF CONTENTS .....</b>	<b>1</b>
<b>INTRODUCTION .....</b>	<b>2</b>
<b>TURNING AN INVOICE INTO AN ORDERS TABLE.....</b>	<b>3</b>
<b>FIRST NORMAL FORM: NO REPEATING ELEMENTS OR GROUPS OF ELEMENTS .....</b>	<b>5</b>
<b>SECOND NORMAL FORM: NO PARTIAL DEPENDENCIES ON A CONCATENATED KEY .....</b>	<b>7</b>
<b>THIRD NORMAL FORM: NO DEPENDENCIES ON NON-KEY ATTRIBUTES .....</b>	<b>9</b>
<b>REFERENCES FOR FURTHER READING.....</b>	<b>11</b>

# Introduction

This is meant to be a *very* brief tutorial aimed at beginners who want to get a conceptual grasp on the database normalization process. I find it very difficult to visualize these concepts using words alone, so I shall rely as much as possible upon pictures and diagrams.

To demonstrate the main principles involved, we will take the classic example of an **Invoice** and level it to the Third Normal Form. We will also construct an **Entity Relationship Diagram** (ERD) of the database as we go.

**Important Note:** This is *not* a description of how you would actually design and implement a database. The sample database screenshots are not meant to be taken literally, but merely as visual aids to show how the raw data gets shuffled about as the table structure becomes increasingly normalized.

Purists and academics may not be interested in this treatment. I will not cover issues such as the benefits and drawbacks of normalization. For those who wish to pursue the matter in greater depth, a list of references for further reading is provided at the end.

**To begin:** First, memorize the 3 normal forms so that you can recite them in your sleep. The meaning will become clear as we go. Just memorize them for now:

1. No repeating elements or groups of elements
2. No partial dependencies on a concatenated key
3. No dependencies on non-key attributes

## Turning an Invoice into an Orders Table

Consider a typical invoice (**Figure A**). It has been simplified to meet the purposes of this discussion.

Figure A: Invoice

**International Widgets**  
742 Evergreen Terrace  
Springfield, MO

**INVOICE**

INVOICE NO: 125  
DATE: September 13, 2002

**To:** Foo, Inc.  
23 Main St.  
Thorpeburg, TX

Customer No. 56

QUANTITY	ITEM ID	DESCRIPTION	UNIT PRICE	AMOUNT
4	563	56" Blue Freen	3.50	\$14.00
32	851	Spline End (Xtra Large)	.25	\$8.00
5	692	3" Red Freen	12.00	\$60.00
<b>TOTAL DUE</b>				<b>\$82.00</b>

**INVOICE**

INVOICE NO: 126  
September 14, 2002

Customer No. 2

QUANTITY	ITEM ID	DESCRIPTION	UNIT PRICE	AMOUNT
50	750	692	12.00	\$1,750.00
<b>TOTAL DUE</b>				<b>\$9,000.00</b>

This document, or something like it, is the basis of the order fulfillment process of almost any business. Every piece of information you see here is important. How can we capture this information in a database?

First, we can just dump all the data from a couple of invoices into our favorite database program (here I am using Microsoft Access). The result might look something like **Figure B** (next page).

Figure B: orders table

order_id	order_date	customer_id	customer_name	customer_address	customer_city	customer_state	item_id	item_description	item_qty	item_price	item_total_price	order_total_price
125	2002-09-13	56	Foo, Inc	23 Main St.	Thorpeburg	TX	563	56" Blue Freen	4	\$3.50	\$14.00	\$82.00
125	2002-09-13	56	Foo, Inc	23 Main St.	Thorpeburg	TX	851	Spline End (Xtrs	32	\$0.25	\$8.00	\$82.00
125	2002-09-13	56	Foo, Inc	23 Main St.	Thorpeburg	TX	692	3" Red Freen	5	\$12.00	\$60.00	\$82.00
126	2002-09-14	2	Freens R Us	1600 Pennsylv	Washington	DC	563	56" Blue Freen	500	\$3.50	\$1,750.00	\$10,750.00
126	2002-09-14	2	Freens R Us	1600 Pennsylv	Washington	DC	692	3" Red Freen	750	\$12.00	\$9,000.00	\$10,750.00
0		0					0		0	\$0.00	\$0.00	\$0.00

The underlying table structure can be represented as **Figure C**

Some things to notice about the orders table...

- **We didn't call it the *invoices* table.** That's because of the difference between the real-world object called an "Invoice" and the database structures that we are going to create to represent it. This is not a hard-and-fast rule — there is nothing to stop you from naming the table *invoices* if you want — this is just my personal preference.
- **There is no primary key.** You'll notice that the **order\_id** field repeats for each line on the invoice. This repetition disqualifies it for use as a primary key, which must be unique. Don't worry about the lack of a primary key. We would never create a table like this in the real world (at least not at this stage). We're only doing this to make the next point patently obvious—
- **There is a lot of duplicated data here.** To remedy this, we begin the normalization process...

Figure C: orders table structure

orders
order_id
order_date
customer_id
customer_name
customer_address
customer_city
customer_state
item_id
item_description
item_qty
item_price
item_total_price
order_total_price

## First Normal Form: No Repeating Elements or Groups of Elements

The most obvious repeating data here is the Customer information. However, we are not going to discuss that until we get to the Third Normal Form.

The First Normal Form refers specifically to the "item" columns that are repeated in **Figure B**: some inventory items are shared by more than one order. This situation has the potential to make the orders table enormous.

The remedy for this obvious defect is to take these item columns and split them out into their own table, which we call **order\_items**:

**Figure D: orders and order\_items tables**

The screenshot shows two database tables. The first table, 'orders', has columns: order\_id, order\_date, customer\_id, customer\_name, customer\_address, customer\_city, customer\_state, and order\_total\_price. It contains three records. The second table, 'order\_items', has columns: order\_id, item\_id, item\_description, item\_qty, item\_price, and item\_total\_price. It contains six records, showing items associated with the orders from the first table.

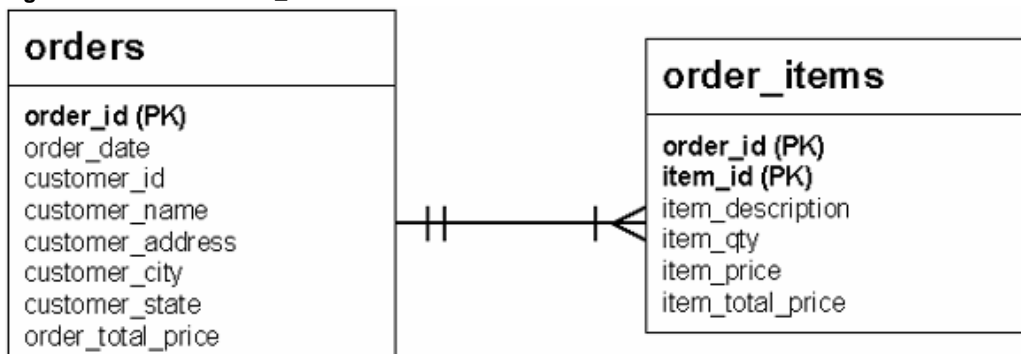
order_id	order_date	customer_id	customer_name	customer_address	customer_city	customer_state	order_total_price
125	2002-09-13	56	Foo, Inc	23 Main St.	Thorpleburg	TX	\$82.00
126	2002-09-14	2	Freens R Us	1600 Pennsylvania A	Washington	DC	\$10,750.00
0		0					\$0.00

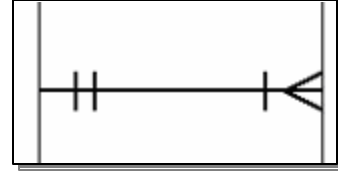
order_id	item_id	item_description	item_qty	item_price	item_total_price
125	563	56" Blue Freen	4	\$3.50	\$14.00
125	851	Spline End (Xtra Large)	32	\$0.25	\$8.00
125	692	3" Red Freen	5	\$12.00	\$60.00
126	563	56" Blue Freen	500	\$3.50	\$1,750.00
126	692	3" Red Freen	750	\$12.00	\$9,000.00
0	0		0	\$0.00	\$0.00

And here is the table structure (**Figure E**):

**Figure E: orders and order\_items table structure**



If you are new to Entity Relationship Diagrams, pay close attention to the line that connects these two tables. This line means, in English,



- each order can be associated with any number of order-items, but **at least** one;
- each order-item is associated with one order, and **only** one.

Don't be dismayed if the meaning of this is not readily apparent at first. For some people (myself included), it is hard to wrap the mind around the notion of an "order-item". Think of an order-item as a single row on the original paper invoice.

An important thing to notice about the **orders** table is that each value in the **order\_id** column is now *unique*. This column now becomes the **primary key** of the orders table. We indicate this with the **(PK)** notation.

Notice that there is no single number that uniquely identifies a row in the order\_items table, like there is for the orders table. Instead, two numbers together uniquely identify each row: **order\_id** and **item\_id**. Both of these numbers together form the primary key for the order\_items table. Even though they are in two different table columns, they are treated as a single entity. We call them **concatenated**.

A value that uniquely identifies a row is called a **primary key**.

When this value is made up of two or more columns, it is referred to as a **concatenated primary key**.

There is nothing to prevent you from using letters as part of a primary key, but it is frequently more convenient to use numbers.

So now our database schema is in First Normal Form. What's next?

#### **Tip: Concatenated Primary Keys**

A real-world example of a concatenated primary key is a 10-digit North American telephone number. The first 3 digits constitute the primary key from an imaginary "Area Code" table, while the remaining 7 digits uniquely identify each telephone subscriber within a particular area code. When put together (or "concatenated"), these 10 numbers uniquely identify any telephone subscriber in the U.S., Canada or the Caribbean.

## Second Normal Form: No Partial Dependencies on a Concatenated Key

Next we test each table for "partial dependencies on a concatenated key". This means that for a table that has a concatenated primary key (and the **orders** table doesn't have one, so we can ignore it here), each column in the table (i.e. each column that is not part of the primary key) *must* depend upon the *entire* concatenated key for its existence. If any column only depends upon one part of the concatenated key, then we say that the entire table has failed Second Normal Form and we must create another table to rectify the failure.

Still not clear? To try and understand this, let's take apart the **order\_items** table column by column. For each column we will ask the question,

"Can this column exist **without** one or the other part of the concatenated primary key?"

If the answer is "yes" — even once — then the table fails Second Normal Form.

Refer to **Figure F**, to remind us of the **order\_items** table structure.

First, recall the meaning of the two columns in the primary key:

- **order\_id** identifies the corresponding row in the **orders** table. It refers to the *invoice* that this item comes from.
- **item\_id** is the inventory item's unique identifier. You can think of it as a part number, inventory control number, SKU, or UPC code.

Figure F:

order_items	
order_id (PK)	
item_id (PK)	
item_description	
item_qty	
item_price	
item_total_price	

Now consider the remaining columns...

**item\_description** is the plain-language description of the inventory item. Obviously it relies on **item\_id**. But can it exist without an **order\_id**?

Yes! A widget could sit on a warehouse shelf forever, and never be purchased. So voilà, our table has failed Second Normal Form already. But let's continue with testing the other columns, to see what we can find out.

**item\_qty** refers to the number of items purchased on a particular invoice. Can this quantity exist without an **item\_id**? Impossible: we cannot talk about the "amount of nothing" (at least not in database design). Can the quantity exist without an **order\_id**? No: a quantity that is purchased with an invoice is meaningless without an invoice. So this column does not violate Second Normal Form (we'll see what we do with such columns in a moment).



*Second Normal Form:  
No Partial Dependencies on a Concatenated Key*

**item\_price** is similar to **item\_description**. It relies on the **item\_id** but not on the **order\_id**, so it *does* violate Second Normal Form.

**item\_total\_price** is similar to **item\_qty**. It relies on both **item\_id** and on **order\_id**, so it *doesn't* violate Second Normal Form.

What do we do with a table that fails Second Normal Form, as this one has? We take the columns that failed the test, and make a new table out of them. We call this new table **items**:

Figure G: order\_items and items table

The image shows two database table windows. The 'order\_items' table has columns: order\_id, item\_id, item\_qty, and item\_total\_price. It contains 5 records. The 'items' table has columns: item\_id, item\_description, and item\_price. It contains 4 records, with the last one being a zero-price item.

order_id	item_id	item_qty	item_total_price
125	563	4	\$14.00
125	851	32	\$8.00
125	692	5	\$60.00
126	563	500	\$1,750.00
126	692	750	\$9,000.00
0	0	0	\$0.00

item_id	item_description	item_price
563	56" Blue Freen	\$3.50
692	3" Red Freen	\$12.00
851	Spline End (Xtra Large)	\$0.25
0		\$0.00

Notice how the columns that didn't violate Second Normal Form stay where they are in the order\_items table.

And here is how the **items** table fits into the overall database schema:

Figure H:



The line that connects the **items** and **order\_items** tables means the following:

- Each item can be associated with *any number* of lines on any number of invoices, including zero;
- each order-item is associated with one item, and **only** one.

We call the **order\_items** table an *associative entity* because it associates, or connects, two other real-world entities (orders and items).

At this point, we have succeeded in attaining Second Normal Form.

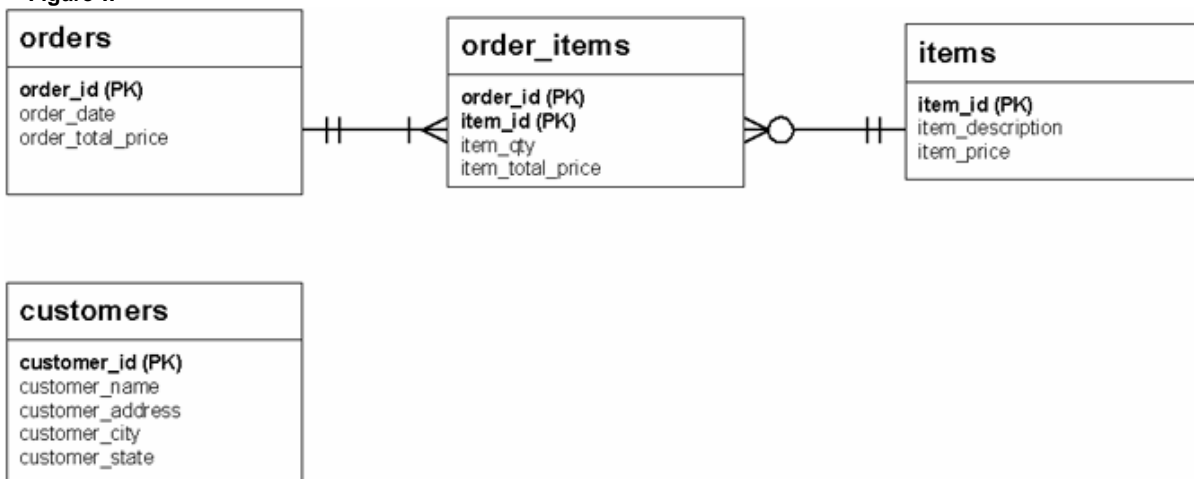
## Third Normal Form: No Dependencies on Non-Key Attributes

At last, we return to the problem of the repeating Customer information. As our database now stands, if a customer places more than one order then we have to input all of that customer's contact information again. This is because there are columns in the **orders** table that rely on "non-key attributes".

To better understand this concept, consider a column like **order\_date**. Can it exist independent of the **order\_id** column? *No*: an "order date" is meaningless without an order. Similarly, **order\_total\_price** cannot exist without **order\_id**. Both **order\_date** and **order\_total\_price** are said to *depend on a key attribute*.

What about **customer\_name** — can it exist on its own, outside of the **orders** table? Yes. It is meaningful to talk about a customer name without referring to an order or invoice. The same goes for **customer\_address**, **customer\_city**, and **customer\_state**. These four columns actually rely on **customer\_id**. They belong in their own table, with **customer\_id** as the primary key (see **Figure I**).

Figure I:

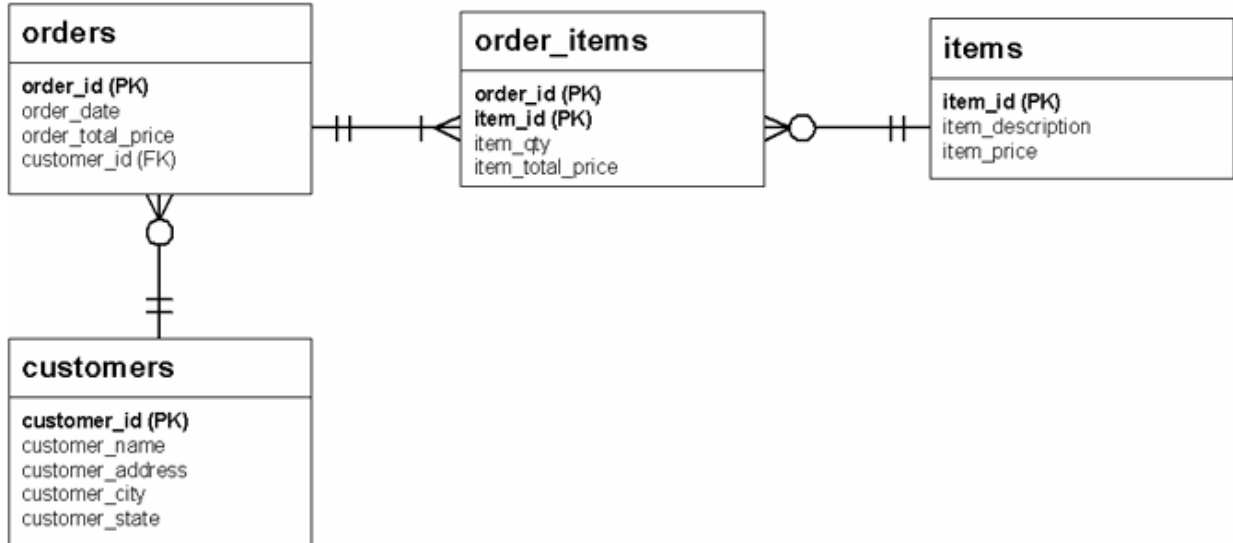


However, you will notice in Figure I that we have severed the relationship between the **orders** table and the Customer data that used to inhabit it.

This won't do at all. We have to restore the relationship by creating an entity called a **foreign key** (indicated in our diagram by **(FK)**). This is essentially a column that points to the primary key in another table. **Figure J** describes this relationship, and shows our completed ERD:

*Third Normal Form:  
No Dependencies on Non-Key Attributes*

Figure J:



The relationship that has been established between the **orders** and **customers** table may be expressed in this way:

- each order is made by one, and **only** one customer;
- each customer can make any number of orders, including zero.

And finally, here is what the data in each of the four tables looks like. Notice that this time, we did not remove any rows from any of the tables like the previous two Normal Forms did.

Figure K:

**orders : Table**

order_id	customer_id	order_date	order_total_price
125	56	2002-09-13	\$82.00
126	2	2002-09-14	\$10,750.00
0	0		\$0.00

Record: 2 of 2

**order\_items : Table**

order_id	item_id	item_qty	item_total_price
125	563	4	\$14.00
125	851	32	\$8.00
125	692	5	\$60.00
126	563	500	\$1,750.00
126	692	750	\$9,000.00
0	0	0	\$0.00

Record: 5 of 5

**customers : Table**

customer_id	customer_name	customer_address	customer_city	customer_state
2	Freens R Us	1600 Pennsylvania Ave.	Washington	DC
56	Foo, Inc	23 Main St.	Thorpeburg	TX
0				

Record: 2 of 2

**items : Table**

item_id	item_description	item_price
563	56" Blue Freen	\$3.50
692	3" Red Freen	\$12.00
851	Spline End (Xtra Large)	\$0.25
0		\$0.00

Record: 3 of 3

## References for Further Reading

Needless to say, there's a lot more to it than this. If you want to read more about the theory and practice of the 3 Normal Forms, here are some suggestions:

- [The Art of Analysis](#), by Dr. Art Langer, devotes considerable space to normalization. Springer-Verlag Telos (January 15, 1997)  
ISBN: 0387949720
- Dr. Codd's seminal 1969 paper on database normalization:  
[www.acm.org/classics/nov95](http://www.acm.org/classics/nov95)
- The [Wikipedia article on normalization](http://en.wikipedia.org/wiki/Database_normalization) discusses all five normal forms:  
[en.wikipedia.org/wiki/Database\\_normalization](http://en.wikipedia.org/wiki/Database_normalization)