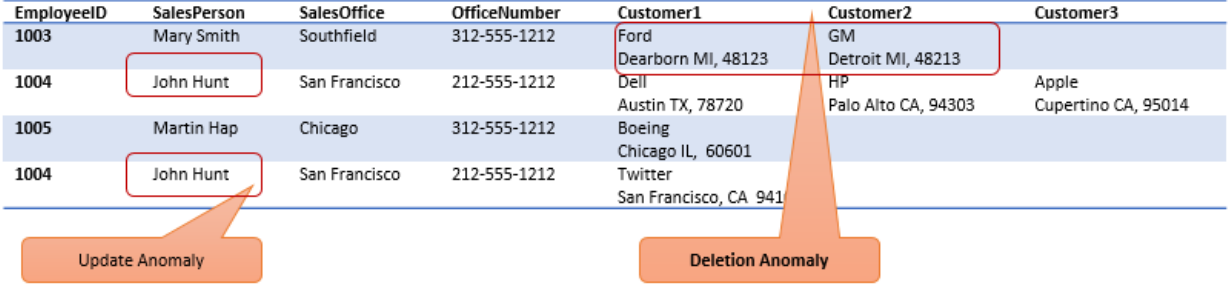
Five Minute Guide to Database Normalization

For this guide we'll use the following as our sample data. This is pretty close to what someone may give you if they were keeping track of information in a spreadsheet.

If you haven't realized it yet, **people + spreadsheets = bad news** for the DB!

Sample Data
Contains data about the sales person, assigned sales office, and their customer contacts.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Southfield	312-555-1212	Ford Dearborn MI, 48123	GM Detroit MI, 48213	
1004	John Hunt	San Francisco	212-555-1212	Dell Austin TX, 78720	HP Palo Alto CA, 94303	Apple Cupertino CA, 95014
1005	Martin Hap	Chicago	312-555-1212	Boeing Chicago IL, 60601		
1004	John Hunt	San Francisco	212-555-1212	Twitter San Francisco, CA 941		



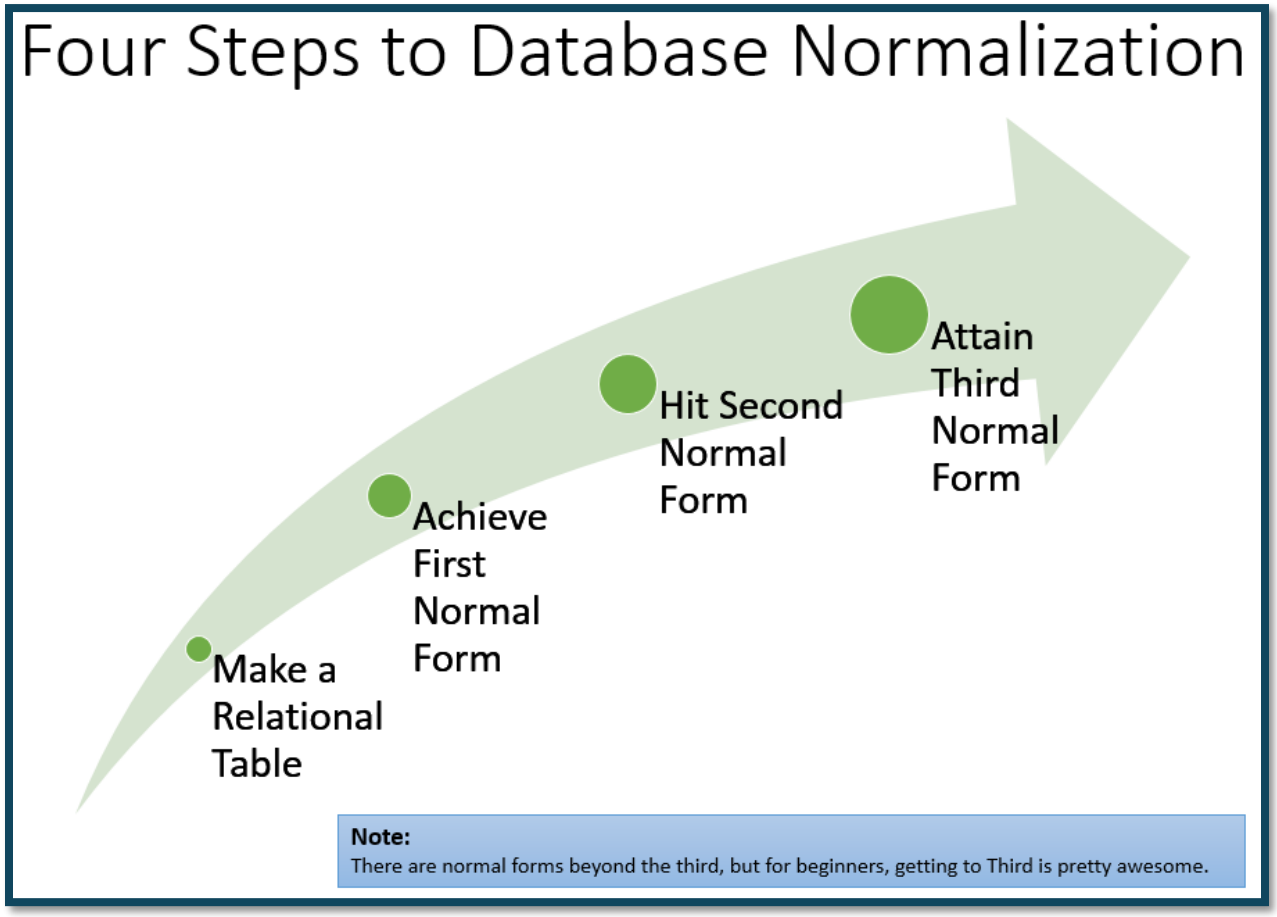
Update Anomaly
Why should I have to change John's name twice? Think about a really large database! It is inefficient to have to change many records just to change a name...

Deletion Anomaly
Just because I delete Mary's records doesn't mean I want to loose customer information for Ford and GM. Actions to one entity shouldn't necessarily affect another...

There are many reasons to normalize data, two are given above. Most reasons hinge around performance or data modification issues.

Data normalization is a two edged knife. When you normalize a database, the data is split into many tables. Eventually that data must be reconstructed for a human to make sense of it...

In this guide I'll go four main steps to take our sales data all the way form it unstructured format to the third normal form. Along the way I'll show you the problems and what we have done to fix them.



The first step is to make a relation table. Let's get started!

MAKE A RELATIONAL TABLE

The first step is pretty easy. All we need to ensure is that each row in our table can be uniquely identified.

You can use one or more column values to uniquely identify a table's row, but most designers settle on one column.

Characteristics of a Relational Table

- Made up of rows and columns.
- Rows are uniquely identified by one or more columns, this is the primary key.

What is Wrong?

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Southfield	312-555-1212	Ford Dearborn MI, 48123	GM Detroit MI, 48213	
1004	John Hunt	San Francisco	212-555-1212	Dell Austin TX, 78720	HP Palo Alto CA, 94303	Apple Cupertino CA, 95014
1005	Martin Hap	Chicago	312-5	601		
1004	John Hunt	San Francisco	212-	CA 94101		

Where is the primary key? Rows are not uniquely identified

Fixed!

Employee EntryID	EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
E100	1003	Mary Smith	Southfield	312-555-1212	Ford Dearborn MI, 48123	GM Detroit MI, 48213	
E200	1004	John Hunt	San Francisco	212-555-1212	Dell Austin TX, 78720	HP Palo Alto CA, 94303	Apple Cupertino CA, 95014
E300	1005	Martin Hap	Chicago	312-555-1212	Boeing Chicago IL, 60601		
E250	1		San Francisco	212-555-1212	Twitter San Francisco, CA 94101		

Added Primary Key

Benefit:
We can now identify each row. This becomes critical once we create more tables and need to relate one to another.

In our examples we'll underline the column names to indicate the primary key.

ACHIEVE FIRST NORMAL FORM

You notice that the rules for database normalization build upon one another.

The first rule for first normal form is that the table in question is a relational table.

First Normal Form
- Data is in a relational table
- Columns contain atomic values
- There are no repeating column groups

What is Wrong?

Customer 1, 2, and 3 is a repeating column group

Employee EntryID	EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
E100	1003	Mary Smith	Southfield	312-555-1212	Ford Dearborn MI, 48123	GM Detroit MI, 48213	
E200	1004	John Hunt	San Francisco	212-555-1212	Dell Austin TX, 78720	HP Palo Alto CA, 94303	Apple Cupertino CA, 95014
E300	1005	Martin Hap	Chicago	312-555-1212	Boeing Chicago IL, 60601		
E250	1004	John Hunt	San Francisco	212-555-1212	Twitter San Francisco, CA 9		

Data is not atomic. Each column has four values: company, city, state, and postal code

Fixed!

Employee EntryID	EmployeeID	SalesPerson	SalesOffice	OfficeNumber
E100	1003	Mary Smith	Southfield	312-555-1212
E200	1004	John Hunt	San Francisco	212-555-1212
E300	1005	Martin Hap	Chicago	312-555-1212
E250	1004	John Hunt	San Francisco	212-555-1212

Repeated columns are now separated rows

Employee ID	Customer ID	Customer	City	State	Zip
1003	C100	Ford	Dearb n	MI	48123
1003	C200	GM	Detroit	MI	48123
1004	C300	Dell	Austin	TX	78720
1004	C400	HP	Palo Alto	CA	94303
1004	C500	Apple	Cupertino	CA	95014
1005	C600	Boeing	Chicago	IL	60601
1004	C700	Twitter	San Francisco	CA	94101

Benefit:
Data is easier to sort and search.

By eliminating repeating column groups we not only start to remove some of the update and delete anomalies we saw in the beginning, but start to make it much easier to sort the data.

Imagine sorting the first table by customer name… how would you go about doing that?

HIT SECOND NORMAL FORM

In the example below, the primary key is the combination of the EmployeeID and CustomerID. The issue here is that if we no longer want to associate an employee with a particular customer, then deleting that row may wipe out all the data for the customer. This is a deletion anomaly.

Second Normal Form

- The table is in 1st normal from.
- All non-key columns are dependent on the table's primary key

What is Wrong?

Customer Information isn't dependent on Employee ID (part of primary key)

Employee ID	Customer ID	Customer	City	State	Zip
1003	C100	Ford	Dearborn	MI	48123
1003	C200	GM	Detroit	MI	48123
1004	C300	Dell	Austin	TX	78720
1004	C400	HP	Palo Alto	CA	94303
1004	C500	Apple	Cupertino	CA	95014
1005	C600	Boeing	Chicago	IL	60601
1004	C700	Twitter	San Francisco	CA	94101

Fixed!

Employee ID	Customer ID
1003	C100
1003	C200
1004	C300
1004	C400
1004	C500
1005	C600
1004	C700

Customer ID	Customer	City	State	Zip
C100	Ford	Dearborn	MI	48123
C200	GM	Detroit	MI	48123
C300	Dell	Austin	TX	78720
C400	HP	Palo Alto	CA	94303
C500	Apple	Cupertino	CA	95014
C600	Boeing	Chicago	IL	60601
C700	Twitter	San Francisco	CA	94101

All fields depend on primary key

This is sometimes called an "intersection table" as it defines the many-to-many relationship between Employees and Sales Offices

Benefit:
We are eliminating update, insert, and deletion anomalies related to customers.

ATTAIN THIRD NORMAL FORM

The key to understanding the third normal form figuring out what the heck "Transitive Dependence" means.

If you having trouble, [I have a more elaborate explanation here!]

Third Normal Form

- The table is in 2nd normal from.
- It contains only columns that are non-transitively dependent on the primary key

What is Wrong?

City and State are transitively dependent on Zip

Customer ID	Customer	City	State	Zip
C100	Ford	Dearborn	MI	48123
C200	GM	Detroit	MI	48123
C300	Dell	Austin	TX	78720
C400	HP	Palo Alto	CA	94303
C500	Apple	Cupertino	CA	95014
C600	Boeing	Chicago	IL	60601
C700	Twitter	San Francisco	CA	94101

Fixed!

No transitive dependencies

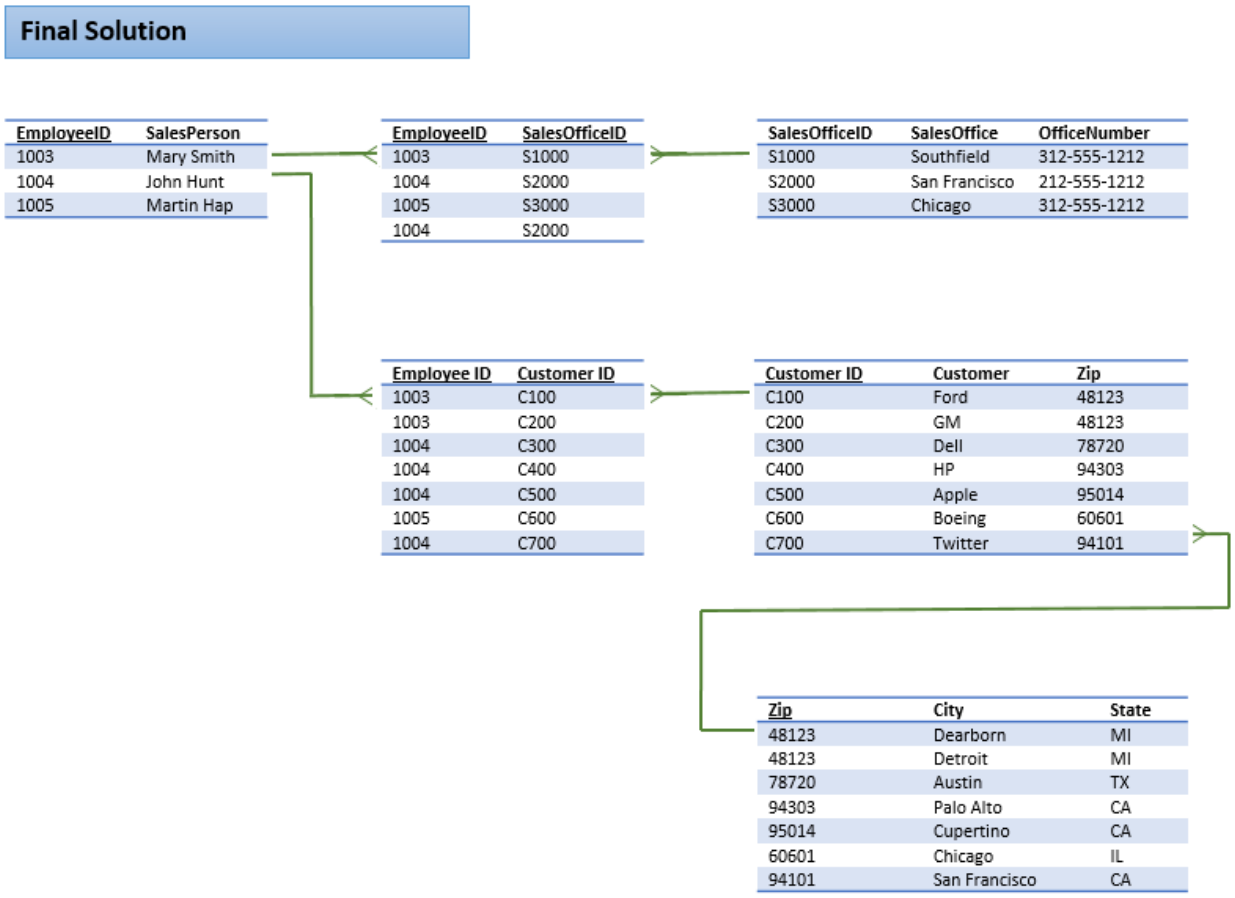
Customer ID	Customer	Zip
C100	Ford	48123
C200	GM	48123
C300	Dell	78720
C400	HP	94303
C500	Apple	95014
C600	Boeing	60601
C700	Twitter	94101

Zip	City	State
48123	Dearborn	MI
48123	Detroit	MI
78720	Austin	TX
94303	Palo Alto	CA
95014	Cupertino	CA
60601	Chicago	IL
94101	San Francisco	CA

Benefit:
We are eliminating update, insert, and deletion anomalies related to customers.

FINAL TABLE LAYOUT

Here is every table in third normal form. The green lines indicate the one to many relationships.



If you have questions or comments please let me know. I would love to help understand this.

If may seem overwhelming at first, but once you get the hang of it, you be able to normalize databases rather quickly.

It kind of like riding a bike, hard to explain, but easy to do with practice.

Enjoy!

Kris Wenzel