Database Second Normal Form Explained in Simple English

This is the third in a series of posts teaching normalization.

The second post focused on the first normal form, its definition, and examples to hammer it home.

Now it is time to take a look at the **second normal form**. I like to think the reason we place tables in 2nd normal form is to narrow them to a single purpose. Doing so bring's clarity to the database design, makes it easier for us to describe and use a table, and tends to eliminate modification anomalies.

This stems from the primary key identifying the main topic at hand, such as identifying buildings, employees, or classes, and the columns, serving to add meaning through descriptive attributes.

An EmployeeID isn't much on its own, but add a name, height, hair color and age, and now you're starting to describe a real person.

So what is the definition of 2nd normal form?

2NF – Second Normal Form Definition

A table is in 2nd Normal Form if:

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

We already know about the 1st normal form, but what about the second requirement? Let me try to explain.

The primary key provides a means to uniquely identify each row in a table. When we talk about columns depending on the primary key, we mean, that in order to find a particular value, such as what color is Kris' hair, you would first have to know the primary key, such as an EmployeeID, to look up the answer.

Once you identify a table's purpose, then look at each of the table's columns and ask yourself, "Does this column serve to describe what the primary key identifies?"

- If you answer "yes," then the column is dependent on the primary key and belongs in the table.
- If you answer "no," then the column should be moved different table.

When all the columns relate to the primary key, they naturally share a common purpose, such as describing an employee. That is why I say that when a table is in second normal form, it has a single purpose, such as storing employee information.

Issues with our Example Data Model

So far we have taken our example to the first normal form, and it has several issues.

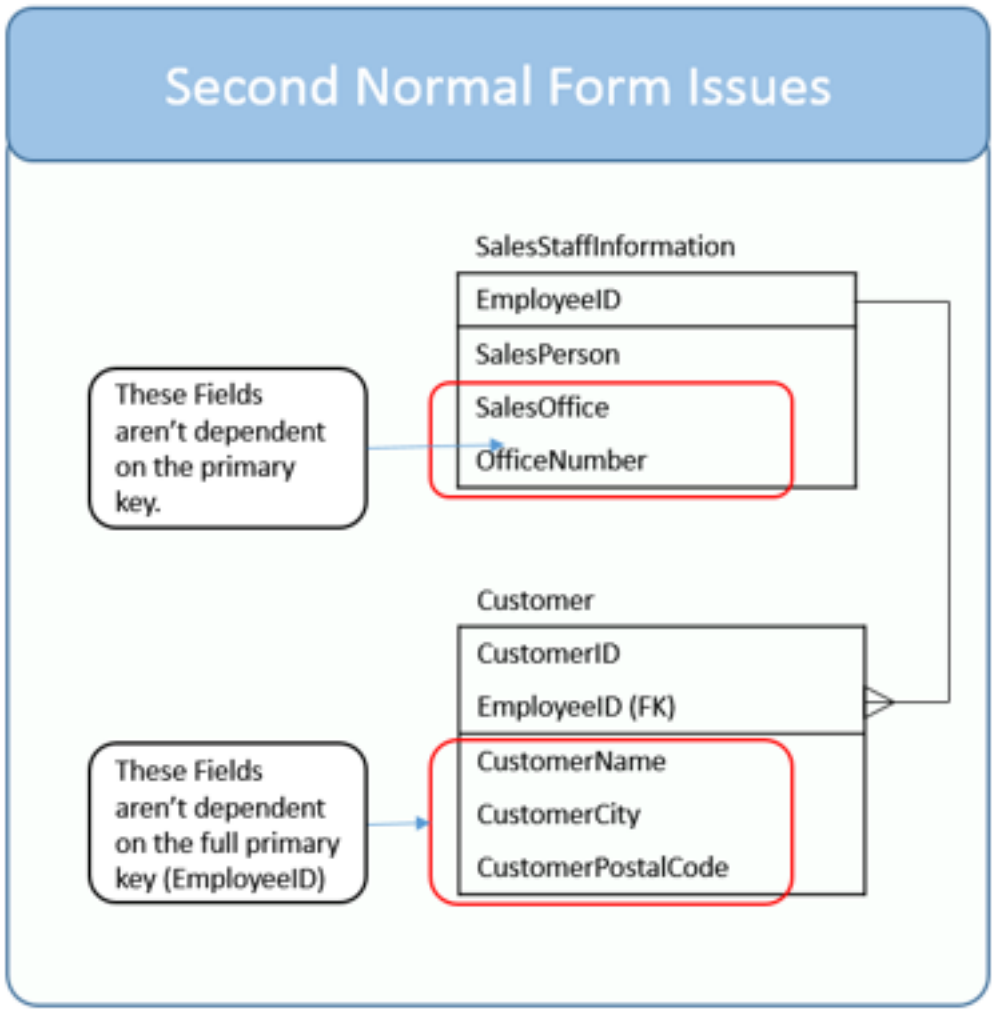
The first issue is the SalesStaffInformation table has two columns which aren't dependent on the EmployeeID. Though they are used to describe which office the SalesPerson is based out of, the SalesOffice and OfficeNumber columns themselves don't serve to describe who the employee is.

The second issue is that there are several attributes which don't completely rely on the entire Customer table primary key. For a given customer, it

doesn't make sense that you should have to know both the CustomerID and EmployeeID to find the customer.

It stands to reason you should only need to know the CustomerID. Given this, the Customer table isn't in 2nd normal form as there are columns that aren't dependent on the full primary key. They should be moved to another table.

These issues are identified below in red.



Fix the Model to 2NF Standards

Since the columns identified in red aren't completely dependent on the table's primary key, it stands to reason they belong elsewhere. In both cases, the columns are moved to new tables.

In the case of SalesOffice and OfficeNumber, a SalesOffice was created. A foreign key was then added to SalesStaffInformaiton so we can still describe in which office a sales person is based.

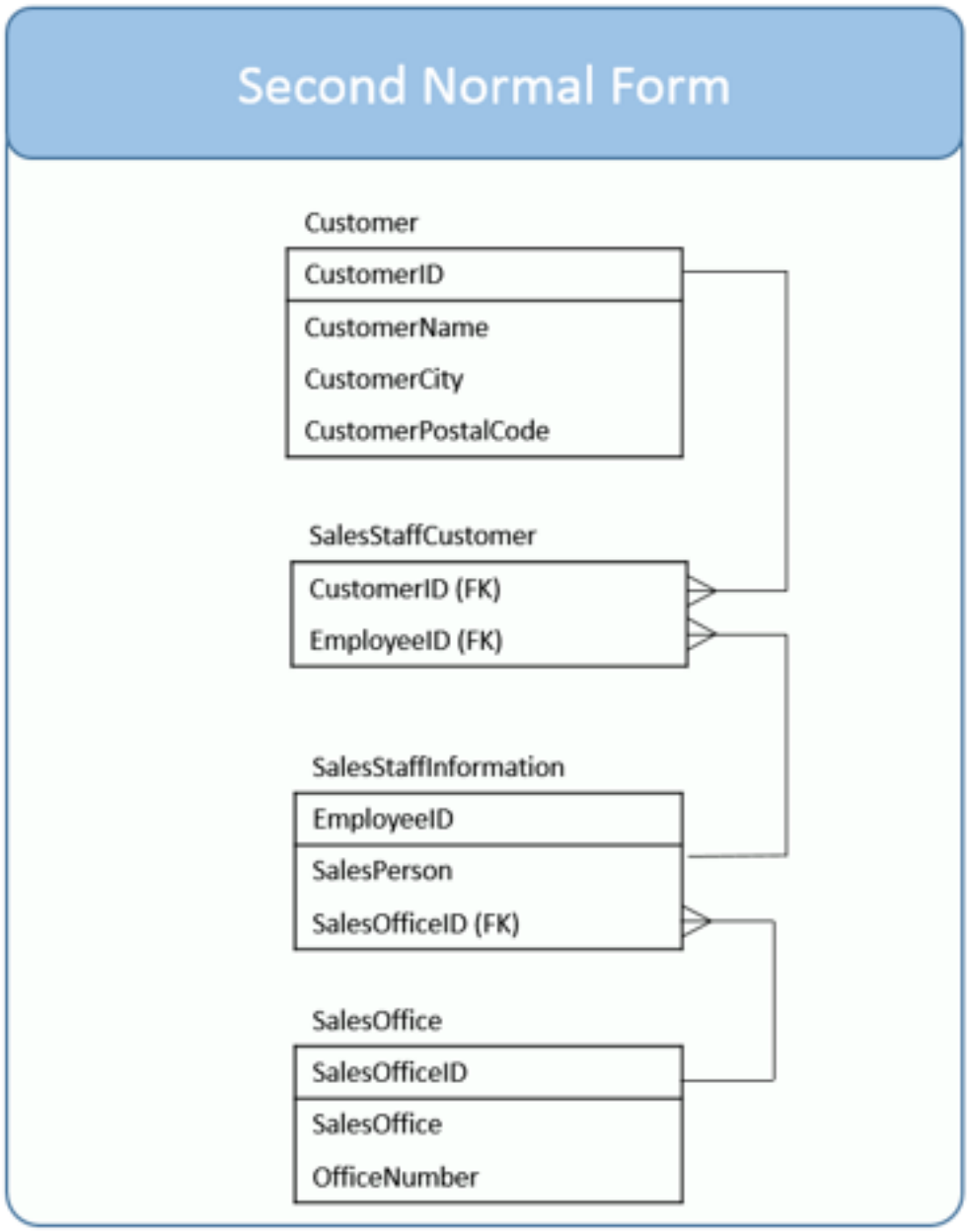
The changes to make Customer a second normal form table are a little trickier. Rather than move the offending columns CustomerName, CustomerCity, and CustomerPostalCode to new table, recognize that the issue is EmployeeID! The three columns don't depend on this part of the key. Really this table is trying to serve two purposes:

1. To indicate which customers are called upon by each employee
2. To identify customers and their locations.

For the moment remove EmployeeID from the table. Now the table's purpose is clear, it is to identify and describe each customer.

Now let's create a table named SalesStaffCustomer to describe which customers a sales person calls upon. This table has two columns CustomerID and EmployeeID. Together, they form a primary key. Separately, they are foreign keys to the Customer and SalesStaffInformation tables respectively.

With these changes made the data model, in second normal form, is shown below.



To better visualize this, here are the tables with data.

Customer			
CustomerID	**CustomerName**	**CustomerCity**	**CustomerPostalCode**
C1000	Ford	Dearborn	48123
C1010	GM	Detroit	48213
C1020	Dell	Austin	78720
C1030	HP	Palo Alto	94303
C1040	Apple	Cupertino	95014
C1050	Boeing	Chicago	60601

As you review the data in the tables notice that the redundancy is mostly eliminated. Also, see if you can find any update, insert, or deletion anomalies. Those too are gone. You can now eliminate all the sales people, yet retain customer records. Also, if all the SalesOffices close, it doesn't mean you have to delete the records containing sales people.

SalesStaffCustomer	
CustomerID	**EmployeeID**
C1000	1003
C1010	1003
C1020	1004
C1030	1004
C1040	1004
C1050	1005

The SalesStaffCustomer table is a strange one. It's just all keys! This type of table is called an intersection table. An **intersection table** is useful when you need to model a many-to-many relationship.

Each column is a foreign key. If you look at the data model you'll notice that there is a one to many relationship to this table from SalesStaffInformation and another from Customer. In effect the table allows you to bridge the two tables together.

SalesStaffInformation		
EmployeeID	**SalesPerson**	**SalesOffice**
1003	Mary Smith	S10
1004	John Hunt	S20
1005	Martin Hap	S10

SalesOffice		
SalesOfficeID	**SalesOffice**	**OfficeNumber**
S10	Chicago	312-555-1212
S20	New York	212-555-1212

For all practical purposes this is a pretty workable database. Three out of the four tables are even in third normal form, but there is one table which still has a minor issue, preventing it from being so.

Check out our last post to learn about the third normal form.

More tutorials are to follow! Remember! I want to remind you all that if you have other questions you want answered, then post a comment or tweet me.

I'm here to help you. What other topics would you like to know more about?

This is the fourth in a series of posts teaching normalization.

The third post focused on the second normal form, its definition, and examples to hammer it home.

Once a table is in second normal form, we are guaranteed that every column is dependent on the primary key, or as I like to say, the table serves a single purpose. But what about relationships among the columns? Could there be dependencies between columns that could cause an inconsistency?

A table containing both columns for an employee's age and birth date is spelling trouble, there lurks an opportunity for a data inconsistency!

How are these addressed? By the **third normal form**.

3NF – Third Normal Form Definition

A table is in third normal form if:

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

Wow! That's a mouthful. What does *non-transitively dependent* mean? Let's break it down.

Transitive

When something is *transitive*, then a meaning or relationship is the same in the middle as it is across the whole. If it helps think of the prefix *trans* as meaning "across." When something is transitive, then if something applies from the beginning to the end, it also applies from the middle to the end.

Since ten is greater than five, and five is greater than three, you can infer that ten is greater than three.

In this case, the greater than comparison is transitive. In general, if **A** is greater than **B**, and **B** is greater than **C**, then it follows that **A** is greater than **C**.

If you're having a hard time wrapping your head around "transitive" I think for our purpose it is safe to think "through" as we'll be reviewing to see how one column in a table may be related to others, *through* a second column.

Dependence

An object has a dependence on another object when it relies upon it. In the case of databases, when we say that a column has a dependence on another column, we mean that the value can be derived from the other. For example, my age is dependent on my birthday. Dependence also plays an important role in the definition of the second normal form.

Transitive Dependence

Now let's put the two words together to formulate a meaning for transitive dependence that we can understand and use for database columns.

I think it is simplest to think of transitive dependence to mean a column's value *relies* upon another column *through* a second intermediate column.

Consider three columns: AuthorNationality, Author, and Book. Column values for AuthorNationality and Author rely on the Book; once the book is known, you can find out the Author or AuthorNationality. But also notice that the AuthorNationality relies upon Author. That is, once you know the Author, you can determine their nationality. In this sense then, the AuthorNationality relies upon Book, via Author. This is a transitive dependence.

This can be generalized as being three columns: A, B and PK. If the value of A relies on PK, and B relies on PK, and A also relies on B, then you can say that A *relies* on PK *though* B. That is A is transitively dependent on PK.

Let's look at some examples to understand further.

Primary Key (PK)	**Column A**	**Column B**	**Transitive Dependence?**
PersonID	FirstName	LastName	No, In Western cultures a person's last name is based on their father's LastName, whereas their FirstName is given to them.
PersonID	BodyMassIndex	IsOverweight	Yes, BMI over 25 is considered overweight.It wouldn't make sense to have the value IsOverweight be true when the BodyMassIndex was < 25.
PersonID	Weight	Sex	No:There is no direct link between the weight of a person and their sex.
VehicleID	Model	Manufacturer	Yes:Manufacturers make specific models. For instance, Ford creates the Fiesta; whereas, Toyota manufacturers the Camry.

To be non-transitively dependent, then, means that all the columns are dependent on the primary key (a criteria for 2nd normal form) and no other columns in the table.

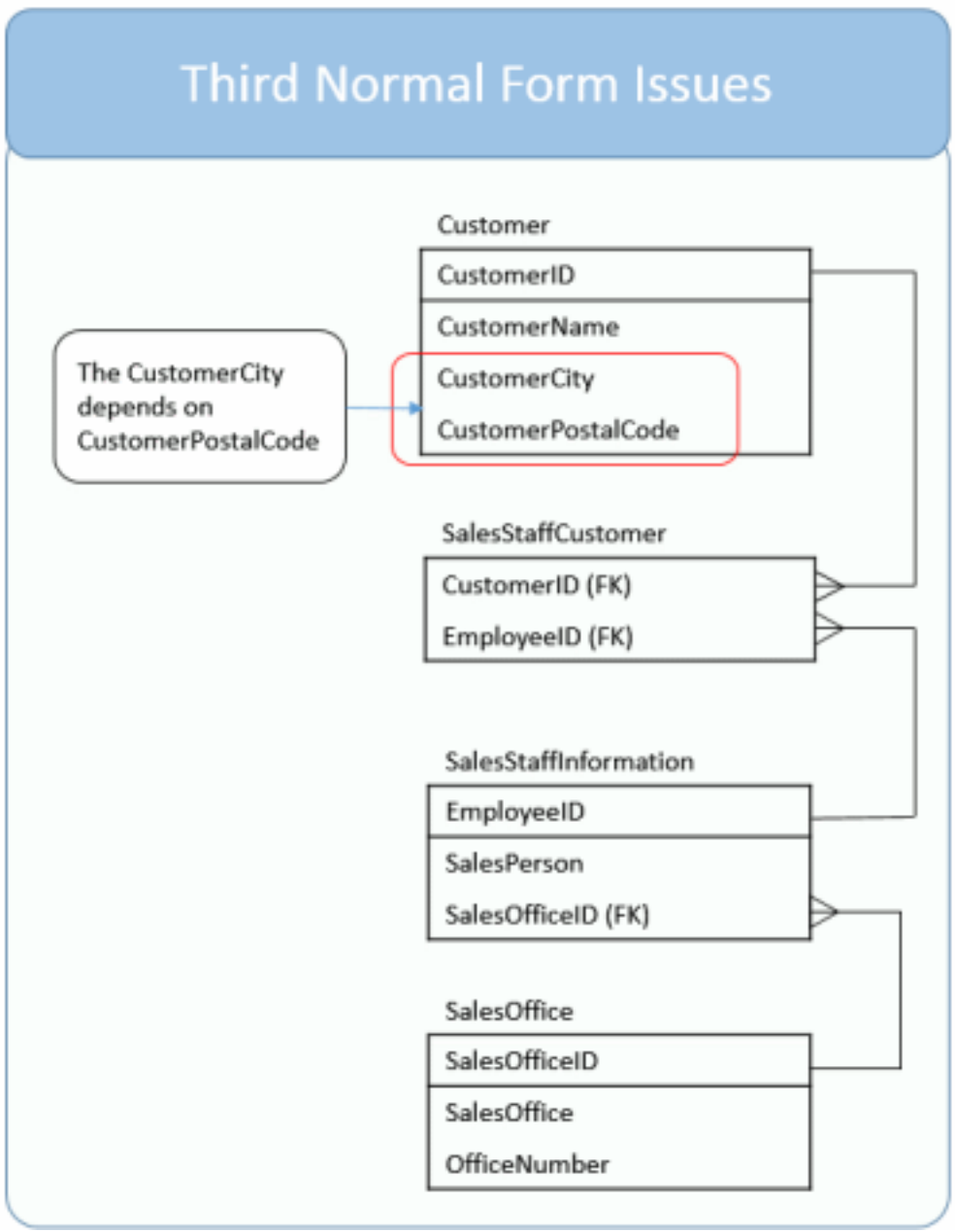
Issues with our Example Data Model

Let's review what we have done so far with our database. You'll see that I've found one transitive dependency:

CustomerCity relies on CustomerPostalCode which relies on CustomerID

Generally speaking a postal code applies to one city. Although all the columns are dependent on the primary key, CustomerID, there is an opportunity for an update anomaly as you could update the CustomerPostalCode without making a corresponding update to the CustomerCity.

We've identified this issue in red.



Fix the Model to 3NF Standards

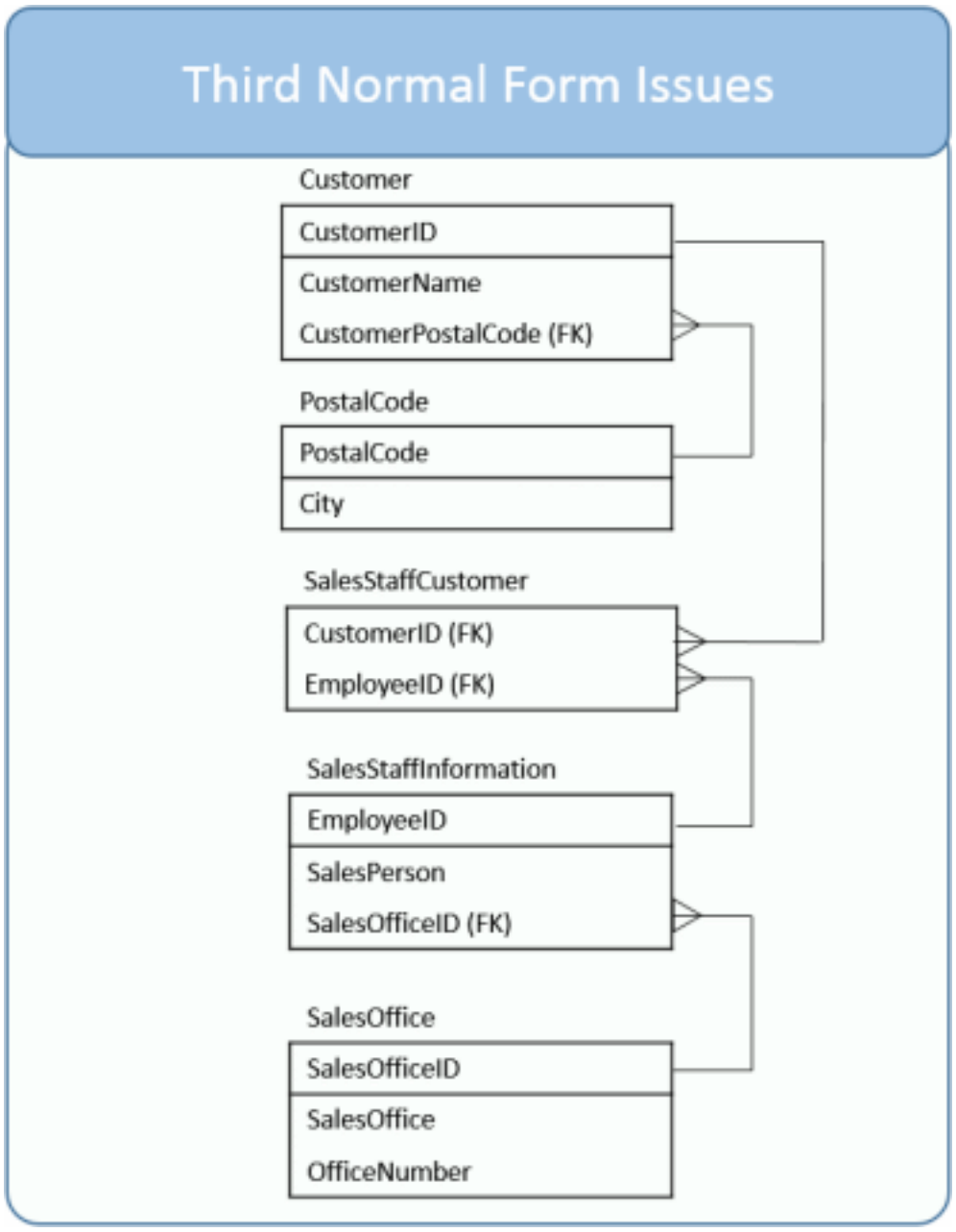
In order for our model to be in third normal form, we need to remove the transitive dependencies. As we stated our dependency is:

CustomerCity relies on CustomerPostalCode which relies on CustomerID

It is OK that CustomerPostalCode relies on CustomerID; however, we break 3NF by including CustomerCity in the table. To fix this we'll create a new table, PostalCode, which includes PostalCode as the primary key and City as its sole column.

The CustomerPostalCode remains in the customer table. The CustomerPostalCode can then be designated a foreign key. In this way, through the relation, the city and postal code is still known for each

customer. In addition, we've eliminated the update anomaly.



To better visualize this, here are the Customer and PostalCode tables with data.

Customer		
CustomerID	**CustomerName**	**CustomerPostalCode**
C1000	Ford	48123
C1010	GM	48213
C1020	Dell	78720
C1030	HP	94303
C1040	Apple	95014
C1050	Boeing	60601

Now each column in the customer table is dependent on the primary key. Also, the columns don't rely on one another for values. Their only dependency is on the primary key.

PostalCode	
PostalCode	**City**
48123	Dearborn
48213	Detroit
60601	Chicago
78720	Austin
94303	Palo Alto
95014	Cupertino

The same holds true for the PostalCode table.

At this point our data model fulfills the requirements for the third normal form. For most practical purposes this is usually sufficient; however, there are cases where even further data model refinements can take place. If you are curious to know about these advanced normalization forms, I would encourage you to read about BCNF (Boyce-Codd Normal Form) and more!

Can Normalization Get out of Hand?

Can database normalization be taken too far? You bet! There are times when it isn't worth the time and effort to fully normalize a database. In our example you could argue to keep the database in second normal form, that the CustomerCity to CustomerPostalCode dependency isn't a deal breaker.

I think you should normalize if you feel that introducing update or insert anomalies can severely impact the accuracy or performance of your database application. If not, then determine whether you can rely on the user to recognize and update the fields together.

There are times when you'll intentionally denormalize data. If you need to present summarized or complied data to a user, and that data is very time consuming or resource intensive to create, it may make sense to maintain this data separately.

Several years ago I developed a large engineering change control system which, on the home page, showed each engineer's the parts, issues, and tasks requiring their attention. It was a database wide task list. The task list was rebuilt on-the-fly in real-time using views. Performance was fine for a couple of years, but as the user base grew, more and more DB resources were being spent to rebuild the list each time the user visited the home page.

I finally had to redesign the DB. I replaced the view with a separate table that was initially populated with the view data and then maintained with code to avoid anomalies. We needed to create complicated application code to ensure it was always up-to-date.

For the user experience it was worth it. We traded off complexity in dealing with update anomalies for improved user experience.

This post concludes our series on normalization. If you want to start from the beginning, click here.

More tutorials are to follow! Remember! I want to remind you all that if you have other questions you want answered, then post a comment or tweet me. I'm here to help you. What other topics would you like to know more about?