We've already seen the process of separating the data into related tables or entities. This is part of normalization. The purpose of normalizing or having normal forms is to have a set of rules for removing anomalies and redundancies in our database. If data is repeated when it shouldn't be, it can cause storage and maintenance problems. So normalizing will also help maintaining of our database. When a table contains data that is unnormalized it might contain data on two or more entities. It could also contain repeating values or columns of data or data that is repeated in two or more rows. There are 7 normal forms and you can apply as you go through the process of normalization. You need to apply them in order starting at the first normal form and then going on. We will be going over just the first 3 for now. Usually your database will be considered normalized at that point. The Boyce Codd through the 6th normal form are applied infrequently and they are beyond the scope of this course.

First normal form states that "The value stored at the intersection of each row and column must be a scalar value, and table must not contain any repeating columns. Scalar means a single value that might belong to a number of fields, but itself is one value. The table should also have a primary key

Order#	Cust#	Last	First	Address	Zip	Order_date	Products
05	123	Smith	Sue	60 Main	83406	12/14/19	2 bracelets, necklace, earrings
06	124	Thomas	Tom	99 South St.	84304	01/05/20	ring, watch
07	123	Smith	Sue	60 Main	83406	01/20/20	tiara

This table doesn't follow 1NF because the products field has multiple values. We can also see that there is probably more than one entity involved here as well. At this point we might think well I have a primary key (or maybe a composite key) .Let's focus on solving part of the two problems here—the multiple values in the products column and we will look at the primary key problem as we go from there.

Order#	Cust#	Last	First	Address	Zip	Order_date	Product1	Product2	Product3
05	123	Smith	Sue	60 Main	83406	12/14/19	2 bracelets	necklace	earrings
06	124	Thomas	Tom	99 South St.	84304	01/05/20	ring	watch	null
07	123	Smith	Sue	60 Main	83406	01/20/20	tiara	null	null

What if we just separate those products into their own column? Are we now in first normal form? No, remember it states also that "The table must not contain any repeating columns" We are repeating the products column header basically again and again. How can we predict how many product columns we should have. If a customer orders a hundred different products should be have 100 columns ready to go? Then if another person only orders one product, they'd have 99 columns with a null value. It just doesn't make sense.

Order#	Cust#	Last	First	Address	Zip	Order_date	Products
05	123	Smith	Sue	60 Main	83406	12/14/19	2 bracelets
05	123	Smith	Sue	60 Main	83406	12/14/19	necklace
05	123	Smith	Sue	60 Main	83406	12/14/19	earrings
06	124	Thomas	Tom	99 South St.	84304	01/05/20	ring
06	124	Thomas	Tom	99 South St.	84304	01/05/20	watch
07	123	Smith	Sue	60 Main	83406	01/20/20	tiara

Now we can see we are starting to get it into 1NF. There is only one value in each cell and no column values for products are repeating. But we can see a problem because now some of the data in each row is repeated and our primary key (even if we used Order# and Cust# as a composite) is not unique. This could be solved in 1NF by separating out the two entities here of order and customer. But let's look at 2NF and see if it will help with this.

Second Normal Form 2NF states that "Every non-key column must depend on the entire primary key. " (this has to do with composite keys where each field in the row, every value that's not part of the primary key, has to rely on composite primary key, relying on both parts of it.)

Even if we somehow could have used Order# and Cust# as a composite key that uniquely identifies each row, each field does not rely on entire composite key; the product and order_date really have no relationship with the customer number. And the customer specific columns aren't dependent on the order number. They can make lots of orders in the future.

So let's break it down to two different entities and that will solve one table—the customer table.

Cust#	Last	First	Address	Zip
123	Smith	Sue	60 Main	83406
124	Thomas	Tom	99 South St.	84304

Order#	Cust#	Order_date	Products
05	123	12/14/19	2 bracelets
05	123	12/14/19	necklace
05	123	12/14/19	earrings
06	124	01/05/20	ring
06	124	01/05/20	watch
07	123	01/20/20	tiara

At this point there are no more composite keys so we don't need to worry about 2NF any more. There are no more composite keys (or in our case we never really had one) so we are done with 2NF. So again these steps could have been a part of 1NF as well and then when we came to 2NF we could have just skipped it because there wouldn't have been a composite key.

So we need to apply the 3NF now. 3NF states that "Every non-key must depend only on the primary key. So a lot like 2NF but looking as dependency on just a single value primary key (not-composite).

The customer table is already in 3rd normal form because every non-key attribute, last, first, address, and zip definitely belong to only that customer. The Order table still has a non-key (products) that doesn't depend on the primary key. Product names don't depend on the order number. We know they are going to be used in lots of different orders. And you could also say that the product name describes a product not an order. Another way to look at it is that products really should be it's own entity as well. We will probably need a description of the product and a price of the product and it wouldn't work to add those onto this row either. They would belong to the particular product not the order. But our order date does depend on that particular order and the foreign key of customer needs to be there to relate the two tables. And again our primary key is repeating which is not following what primary keys do—they need to be unique.

If you see a column that doesn't depend on the primary key, you separate it out into it's own table.

Order#	Cust#	Order_date
05	123	12/14/19
06	124	01/05/20
07	123	01/20/20

Product_id	Name	Quantity
004	bracelet	2
005	necklace	1
006	earrings	1
007	ring	1
008	watch	1
009	tiara	1

So now we have a products table, but the products table is still not in 3NF. The quantity field does not depend on the primary key. Also what is the relationship between these two tables. (1:1, 1:M, or M:M)? It's many-to-many. Orders can have many products and products can belong to many orders. So to resolve that and help put it into 3NF at the same time we are going to create a linking table.

Order#	Cust#	Order_date
05	123	12/14/19
06	124	01/05/20
07	123	01/20/20

OrderLine#	Order#	Product_id	Quantity
1	05	004	2
2	05	005	1
3	05	006	1
4	06	007	1
5	06	008	1
6	07	009	1

Product_id	Name	Price
004	bracelet	25.99
005	necklace	35.99
006	earrings	15.99
007	ring	129.99
008	watch	89.99
009	tiara	1999.99

Now we can place quantity inside the linking table where it makes more sense. Now each table is in 3NF. I did add a field onto products called price which does depend on the product_id so that fits. And you could also have other fields like product description, etc.

Also notice the linking table, I didn't necessarily need a new orderline# field as primary key, I could have used the order# and the product_id as a composite key and that would have worked as well.

The quantity attribute works well here because it has to do with the specific order and the specific product of that order.

The rest of the normal forms are used rarely and the last one is only an abstract idea and not a practical design model. We will not go beyond 3NF in this course.

But 1NF through 3NF does a great job in helping us create a proper database.