## note

### Can Excel produce a schema?

A schema is a diagram showing all of the tables and relationships in a data model.

The schema screen grabs shown in this session were taken from Access 2013's *Relationship* view.

Access 2013 is a tool that can be used to design sophisticated relational databases.

You may wonder why Microsoft didn't include the ability to view a schema in Excel 2013.

This feature does exist in their PowerPivot add-in. At the time of writing, this was only available for professional plus versions of Excel 2013.

The standard Excel 2013 product does not include the ability to view a schema.

# Lesson 6-9: Understand many-to-many relationships

## One-to-one relationships.

Up until now you have only modeled one-to-many relationships.

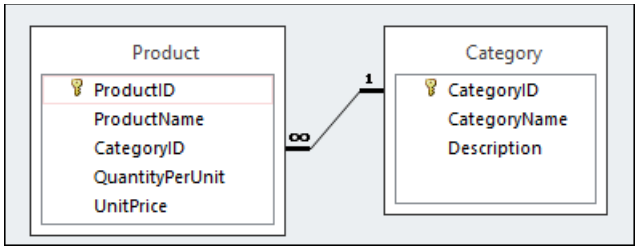
In: *Lesson 6-2: Create a simple data model* you created a relationship between a *Product* table and a *Category* table. This defined the following relationship:

- Each *Product* can have one, and only one, *Category.*
- Each *Category* can potentially be associated with many *Products.*

This type of relationship is called a *one-to-many* relationship with the *Category* on the *one* side of the relationship and the *Product* on the *many* side.

One-to-many relationships are by far the most common type of relationship.

In a database diagram (also called a *schema*) the relationship would be shown like this:




You can see how the one and many side of the relationship are marked. This schema was created using Microsoft Access (see sidebar).

## Many-to-many relationships

A good (and easy to understand) real world many-to-many relationship is found between *Teachers* and *Students* at a school.

- One *Teacher* has many *Students.*
- One *Student* has many *Teachers.*

This type of relationship is more difficult to model than a simple one-to-many relationship.

In the world of commerce, the many-to-many relationship most often encountered is that between *Products* and *Invoices.*

- One *Invoice* can bill many *Products.*
- One *Product* can appear on many *Invoices.*

# note

## Even IT professionals often find many-to-many relationships difficult to understand

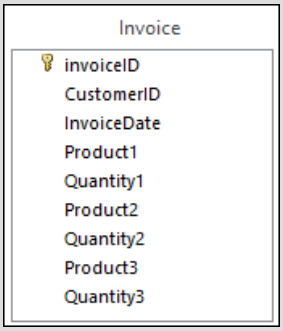
Unfortunately, database design is one of the least understood skills in the world of IT.

I have spent a great deal of my professional life implementing clumsy work-arounds to overcome badly designed databases.

One of the most common (and recurring) errors I have encountered in corporate databases all over the world is the incorrect modeling of many-to-many relationships. Often a programmer that doesn't really understand database design will try to "muddle through".

The normal error is when the programmer didn't "get" many-to-many relationships and tried to model a series of one-to-many relationships instead.

If you see this type of table:




… you will know at once that the designer didn't understand many-to-many relationships.

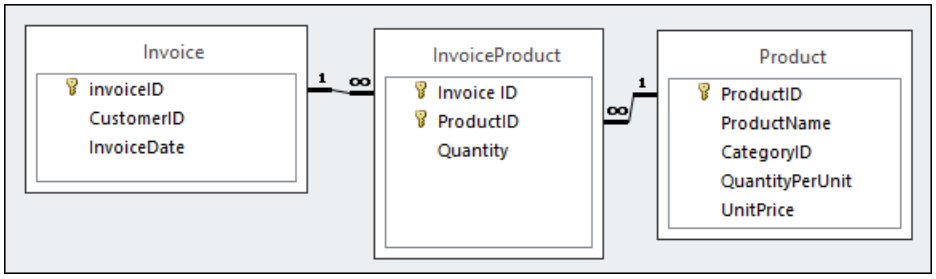
The designer of the above table has tried to work-around the inability to model many-to-many relationships by defining three one-to-many relationships.

You are almost certain to encounter design errors like this if you work with corporate data.

## How many-to-many relationships are modeled

As more fully discussed in: *Lesson 6-1: Understand primary and foreign keys*, a one-to-many relationship uses a primary key at the "one" side of the relationship and a foreign key at the "many" side.

For a many-to-many relationship, a link table is required. This sits between the tables on either side of the many-to-many relationship like this:




The primary key of the link table (*InvoiceProduct*) is a concatenation of the primary keys found in the two related tables.

### *The Invoice table*

This table contains information about the invoice as a whole.

Think of it as the invoice before any items have been added. In this simple example it only contains the *InvoiceDate* and the *InvoiceID* (used in this case as the invoice number). You could also use this to store information such as the ship-to address or due date.

Notice that there is also a *CustomerID* foreign key. This identifies the customer to invoice in a separate *Customer* table (not shown in the schema).

### *The Product table*

This table contains information about each product. In this simple example it only contains *ProductName, QuantityPerUnit* and *UnitPrice* fields. Notice that there is also a *CategoryID* foreign key pointing to the category that the product belongs in (not shown in the schema).

You could also use this table to store other information about the product such as the cost price or wholesale discount.

### *The InvoiceProduct table*

This is the many-to-many link table.

It is a good naming convention to name link tables as a concatenation of the two tables on either side of the many-to-many relationship (in this case the *InvoiceProduct* table joins the *Invoice* and *Product* tables in a many-to-many relationship).

This table defines each line item in the invoice.

In this case we are simply storing the number of units sold on each invoice line. The *ProductID* identifies the product sold on each line of the invoice. This table can be used to obtain the *product name* and *unit price* for each item sold.

© 2017 The Smart Method® Ltd