Microsoft Access® Data Normalizing - A Practical Approach

Jerry Latham, MS MVP, MOS Master

Contents Overview: An explanation of data normalizing is the major part of this document. To help learn at a 'practical' level, a very simple database, cars.mdb, is available to help you visualize how it works. The end of the document takes you through a quick walkthrough on creating forms with subforms using cars.mdb. You will need Access 2000 or 2003 to use the database file.

Introduction: I see many people bravely attempting to move from Excel[1] to Access and I sincerely applaud those efforts. For a long time Excel has been overworked and made to do things that are better handled with a relational database than with a spreadsheet.

The unfortunate side of this is that, through no fault of their own, they are unprepared to deal with designing a smoothly working, efficient database application. They are "stuck" in an Excel mindset and haven't had anyone help them move over to the Access mindset. The bottom line? They haven't had any guidance in the whole purpose and philosophy of relational databases. Hopefully this writing will help with that to some small degree.

Purpose: the whole purpose of a relational database (RDB) is to provide an efficient storage and retrieval system for information. That's it. The fact that you can manage that information and retrieve it in a wide variety of ways to turn it into *useful* information is just a nice side effect of RDB setups.

Example: suppose that the folks over at Merriam-Webster just kept adding words and definitions to their dictionary as they encountered them. And suppose that was all they did. You'd have a ton of potentially useful information but because it was all jumbled up, most people would probably never open the book – too darned hard to find a word. But because the dictionary (database) is ordered and presented alphabetically it becomes *useful* information to us. That is what an RDB does for us: takes information and makes it *useful* information.

Philosophy: The philosophy behind an RDB is that of ***Normalizing***. Cut through all of the icing on that cake and at the core you realize that normalization comes to mean reducing duplicate data entry to an absolute minimum. Beth Melton, MS MVP and MOS Master Instructor has put together an excellent presentation of normalization at the practical level at: Databases: Normalizing Access Data I urge you to read that, it will reinforce what I write about it here.

If you research normalizing you will find that there are 5 levels of normalizing or Normal Form (NF), usually referred to as 1NF, 2NF, 3NF, 4NF and 5NF. The simplest level is 1NF and each of the next levels build on the earlier ones. That is to say that a 5NF data model will also comply with the requirements of the previous four NF levels. I believe that if you read other sources you'll find that many others will tell you that if you reach 3NF that's good enough. Some will disagree strongly. I personally agree that 3NF is 'good enough' for most casual applications. I'll even stick my neck out some and say

[1] Excel and Access are registered trademarks of Microsoft Corporation. Other product names are copyright by their respective owner or owners.

that for many personal or small projects 2NF will meet the "good enough" test. I agree that the better normalized your data is, the better off you are in the long run. However, for most small projects the effort to get to 5NF is often not worth the effort. There's a big difference in criticality of efficiency between the database for the supply system of a worldwide distribution company and that of your personal electronic recipe or rolodex program.

Let us take a quick look at the levels of normalizing and you can judge for yourself how much effort you will need to put into your database efforts. Before we can do that, I need to make sure that we are both on the same page regarding what things like *tables, records, fields, primary key* and *foreign key* are all about.

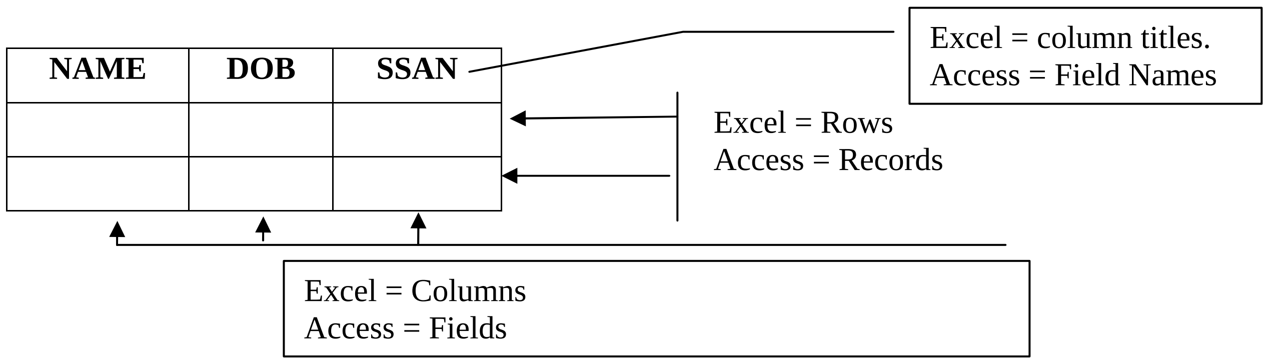
Table – a table in Access can be thought of much as a worksheet in Excel. Ok, I told you to try to get out of the Excel mindset, but I did *not* say to forget all about it!

A table has ***records*** and they are like the rows of information on an Excel worksheet.

Each record has one or more ***fields*** that are like the columns on an Excel worksheet.

Each record (row) contains a group of related information such as a person's name, date of birth (DOB) and social security account number (SSAN). One row deals with just one person's information.

Each one of the data items (name, date of birth, social security number) is a field in the record. Each field should be 'atomic', that is it should only store one piece of information – you shouldn't put Date of Birth and SSAN in the same field, for example.



The First Normal Form (1NF)

To get a database into 1NF you need to consider three elements:

- Eliminate repeated data in individual tables.
- Create separate table(s) for each set of related data
- Identify each set of related data with a primary key

1NF R1 - I just made that up: First Normal Form, Rule 1. Eliminate repeated data in individual tables. This means that there should not be 2 or more rows/records in a table that contain the information about the same person, place or thing. Each record should apply to a unique data group.

1NF R2 – Create separate table(s) for each set of related data. This is actually a place where some people stumble. They have difficulty deciding what “related data” means. Related data is information that is uniquely tied to other entries in the same table. In our example table above, each item of information is related directly to just one person. While we may encounter duplicate names (how many John Jones’ are there in the world), and even duplicate dates of birth (in any group of 366 people or more, guaranteed that at least 2 of them will share a birthday), but we should NOT encounter duplicate social security numbers. So the name is related to the date of birth and both of those are related to the SSAN. Combined they describe one unique person. But if we were to add information such as address and we want to keep up with their home and business addresses, then we would possibly end up with a table with room for two sets of street, city, state and zip codes for them. Technically, the table would still qualify as 1NF with this in it, because each address would be unique to that individual, but it violates rule two in that addresses are made up of “related data”. The related data being information that makes up an address and doesn’t actually have anything to do with identifying a person. We will deal with this in 2NF later.

1NF R3 – identify each set of related data with a ***primary key***. By definition a primary key is unique within the table. No record will have the same primary key as any other record. In this case, because we are including people’s social security numbers, we could use that field as the primary key. Another possible candidate in a business related database would be their account number or customer ID. You may notice that in some company’s files they know you by your phone number which turns out to be unique to their thinking even though every member of your family shares that number. But SSANs are supposed to be truly unique, so it would be a good choice. But there may be a better choice: let the database assign a unique numeric value to each record in a table. Access provides an AutoNumber data type that is superb to use as a primary key.

You may be asking “what good is a primary key – what do you use it for”? The primary key of one table is copied into a field in other tables that contain related information. When it is copied into those related tables, it is known as a ***foreign key*** or ***alien key.*** The key values link the related records in the various tables together.

By using the AutoNumber feature of Access you remove a good chance of errors created during the data entry process. While SSAN should be unique to one person, it may not be if you make a typo when you’re entering it into the system and you may not notice the error until you try to enter information for someone whose SSAN actually turns out to be the one you typed in poorly weeks, months or years ago. So at the 1NF level, our table might look like this:

PrimaryKey	**NAME**	**DOB**	**SSAN**

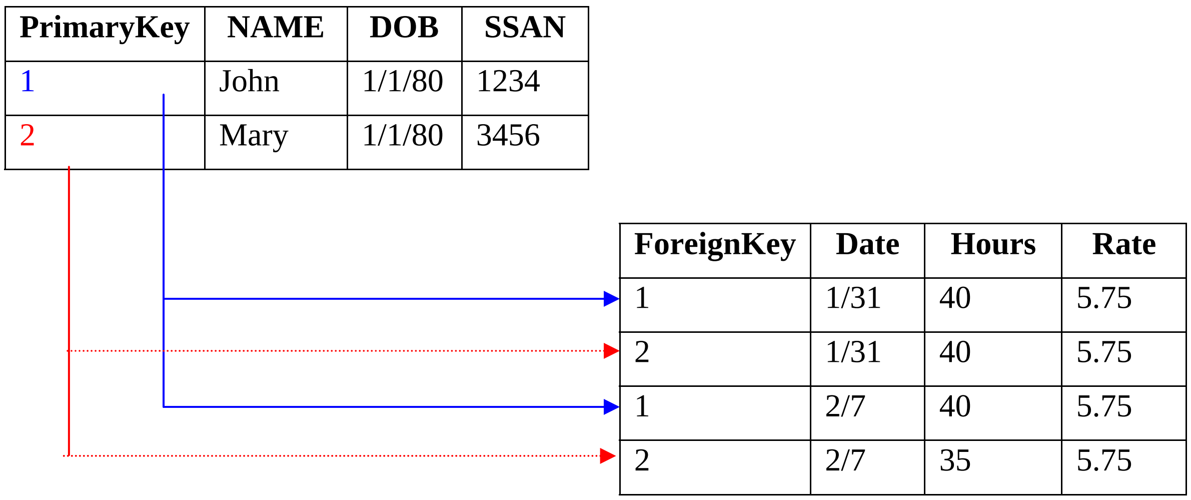
Notice that we’ve added a new field dedicated to holding our Primary Key field for the records in the table. We now have a table that meets the requirements of 1NF.

The Second Normal Form (2NF)

To get to 2NF you only need to consider two rules:

- Create a separate table for sets of values that apply to multiple records.
- Relate the tables using a foreign key

Consider a personnel database for example. The accounting department probably needs to know your name and SSAN along with information from your timesheet in order to cut you a paycheck. Meanwhile the travel department needs that same information in order to make reservations for you for your next business trip. There is no reason for the accounting and travel departments to share any tables at all. But they do both need to be able to identify their records with your name and SSAN. That is done by including a copy of the ***primary key*** from your basic information table as the ***foreign key*** in their tables where records exist that relate to you. I'll try to show that graphically by showing samples of the basic information table and a simple accounting department table:



PrimaryKey	NAME	DOB	SSAN
1	John	1/1/80	1234
2	Mary	1/1/80	3456

ForeignKey	Date	Hours	Rate
1	1/31	40	5.75
2	1/31	40	5.75
1	2/7	40	5.75
2	2/7	35	5.75

In this way you can keep pay records for numerous individuals for a long time and never confuse them. The records in the pay table that are related to records in the basic information table are identifiable by comparing the Primary and Foreign key values.

If anyone is asking why the pay table doesn't include total pay – that would be a waste of space since it can be calculated by multiplying Hours x Rate any time you need it.

On to 3NF…

The Third Normal Form (3NF)

To get to 3NF you only need to consider one rule:

- Eliminate fields that do not depend on the key.

To illustrate this point, lets look at our basic information table again and consider other information that we MIGHT want to have available about a person. Perhaps we need to know what level of education they have achieved – so we can possibly promote them out of that $5.75/hour position. We could simply enter the information into a separate field, but it might not help us. Lets say we're just starting and only have John and Mary as employees and John is a High School Graduate and so is Mary. If we wanted a list of all possible education levels, we can't create it from the available information. Although we could easily provide a list of the education levels of our current employees with what we have.

It would be better to have a separate table that just contains a list of possible education levels attained. It might look like this:

EdLvlPrimaryKey	**EducationLevelAchieved**
1	Non HS Graduate
2	GED or HS Graduate
3	Trade School Cert
4	Some College
5	Bachelors Degree
6	Masters Degree
7	PhD

There are several ways of associating a basic record with an education level: you could just include the EdLvlPrimaryKey as an added field in the basic employee record or you could create a join table that has lists of matched up PrimaryKey values from the basic record and an associated EdLvlPrimaryKey value from this table. Both work fine.

An added benefit of having separate 'control list' or 'look up' tables like this one is that they provide another way of helping to assure data consistency within your database. Suppose that you didn't have this table and were simply typing the education level into a field in the basic information table. Consider the possible ways to indicate a non-high school graduate:

None

No HS

Not HS Grad

and on and on, not to mention that someone might type "no hs" instead of "No HS". That kind of thing makes searching thru the data later more difficult. But if you had a list to choose from that was based on the education level table, then your entries for education level achieved would be consistent and manageable as a separate data group.

What about **4NF** and **5NF**?

While there are possible benefits to continuing to these levels, especially while taking a database course in college, in the real world they can complicate the structure of your database by requiring more and more detailed tables. This can make your system slow to respond to queries for information from it. Complexity also tends to add risk with system maintenance – and maintenance costs are going to be high enough as it is. Trust me. Over the life of a 'system', software maintenance can account for 90% of the total cost of the software, so that huge budget outlay to get it operational turns out to only be the tip of the iceberg.

In point of fact, most databases you'll look at that work well and are generally well designed are quite possibly only going to meet the requirements of 2NF. 3NF at best.

Even the definitions for 4NF and 5NF can get difficult to contend with.

4NF says that if a table is on the "many" side of a one to many relationship (like our example earlier) it cannot be the "many" side of any other one-to-many relationship.

5NF says that if a table has a field with values that are often repeated (like our education levels would be if we typed them in individually) then the values should be replaced with keys that can be related to a look up table. We actually did that. But we may not be compliant with 5NF because we may have skipped over 4NF earlier.

Taken to the extreme, 5NF says if you have a field that could be answered only with TRUE or FALSE or YES or NO that you should have a separate table with just those two entries in it and keep a numeric reference (foreign key) to that table in your main table. In reality, the inefficiency of going to that lookup table hundreds of times instead of just storing the actual True/False/Yes/No information directly in the basic information table.

Alright, the theory is behind us for a while. How do we put it into practice? Where to start?

As an example problem I'm going to choose a project that hopefully will be sufficiently complex to make us think a little, but at the same time be simple enough so that we can get our heads around it and concentrate on the database side of it more than the details of the whatsit we will be dealing with.

How about an automobile customizing database? Most of us understand enough about cars to make this work. At least that's my hope. We won't go into great and gory detail about all possible options, but hopefully we will look at enough for you to get an idea of how thing work and how you need to look at your own database needs.

Cars – you could set up a database with every conceivable combination of options and just let your customers browse through it and pick the one they want. But that would be a fair sized database and your customers may tire of scrolling through the thousands of possible combinations of colors, body styles, engine sizes, transmission types, sound systems, seat covers, etc.

Begin by examining the whole item (car) and think of the things that make a car a car. Or at least those things that make a car a car that your customers can pick from to personalize their choice.

Cars can come in 2-door or 4-door models. Don't argue with me – you may have one with more or less, but go with me on this one. We are the T-Model of the future maker, we have one model, but instead of just having one color, one engine, etc. we do give people some choices.

Cars have color choices.

Cars have engine choices.

Cars have transmission choices.

Cars have different choices for interiors.

To give us a way of sending spam to our potential customers who decide not to buy our fine product today, we start by create a basic potential purchaser record and ask them to enter their:

Name

Address

Phone Number and the all important

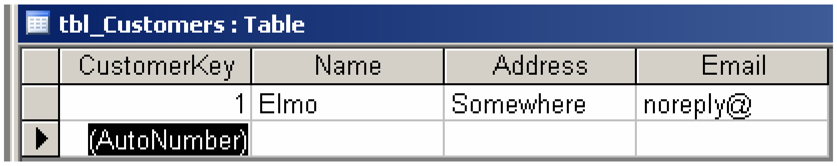
eMail address.

We will need to track what their choices are so we come up with a second table that will hold information about their choice for a car. Why not just include it as part of the basic record? Because they may have a couple of combinations in mind and want printouts of all of them so they can take them home and study them. He may want it in Red and she may prefer Powder Blue.

In this case we would probably create several look up tables, on for each type of choice that can be made.

To start with we make up our two basic related tables:

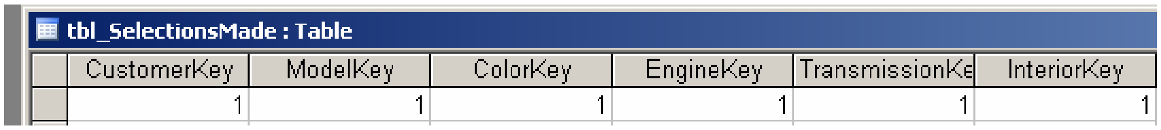
First our potential customers table, with some example data:



tbl_Customers : Table

	CustomerKey	Name	Address	Email
	1	Elmo	Somewhere	noreply@
▶	(AutoNumber)			

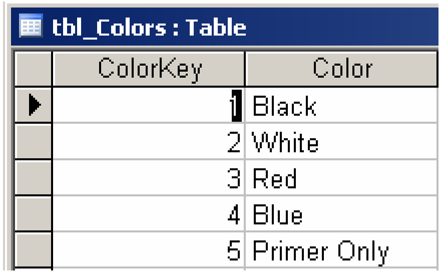
Along with a table to record the choices the potential customer makes:



tbl_SelectionsMade : Table

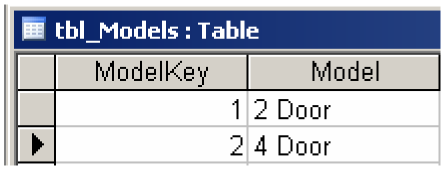
	CustomerKey	ModelKey	ColorKey	EngineKey	TransmissionKe	InteriorKey
	1	1	1	1	1	1

So where did the ‘1’ entries under ModelKey, ColorKey, etc come from? In the table design, I set those as the initial, or default, value for new records. They will change as the customer makes other selections of the features. They are related to primary key fields in our lookup tables. Those tables might look something like these:



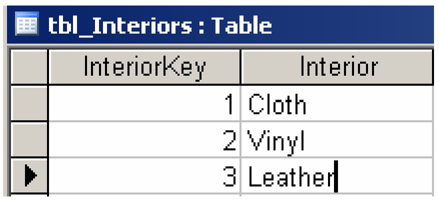
tbl_Colors : Table

	ColorKey	Color
▶	1	Black
	2	White
	3	Red
	4	Blue
	5	Primer Only



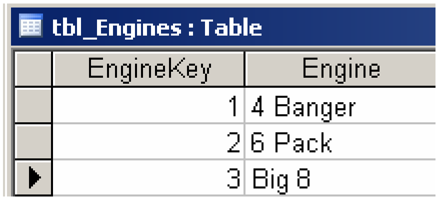
tbl_Models : Table

	ModelKey	Model
	1	2 Door
▶	2	4 Door



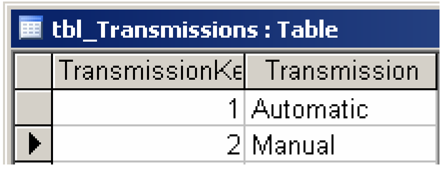
tbl_Interiors : Table

	InteriorKey	Interior
	1	Cloth
	2	Vinyl
▶	3	Leather



tbl_Engines : Table

	EngineKey	Engine
	1	4 Banger
	2	6 Pack
▶	3	Big 8



tbl_Transmissions : Table

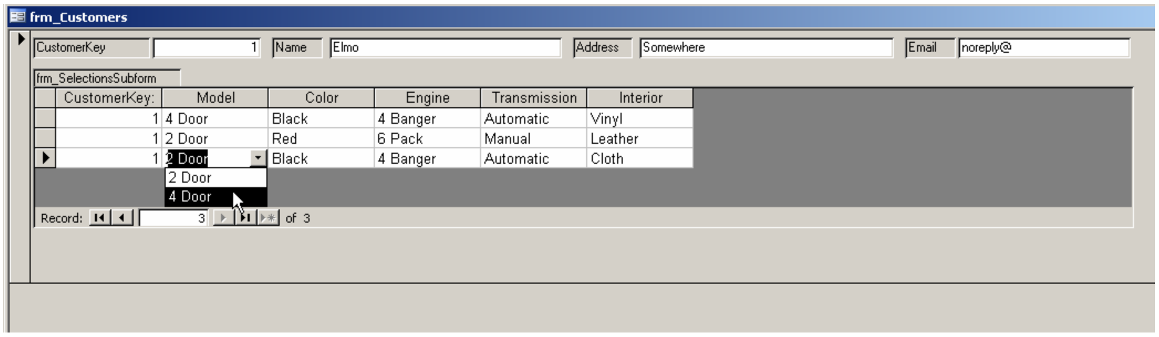
	TransmissionKe	Transmission
	1	Automatic
▶	2	Manual

If we were to ‘look up’ the entries in the various tables based on what we see in the selections key, we see that the default selection is:

A 2-door, black car with cloth interior, a 4-cylinder engine and automatic transmission.

You might ask why store a kind of unintelligible number in the selections made table instead of the plain-English readable choice? Efficiency of storage is the answer. It takes less room in memory to store a number than it does to store a 4 or 5 letter word. Since we are one of the largest dealers in the country with thousands of potential customers visiting our showrooms everyday, every bit of memory or disk space we can save is important to us. But we will change the *appearance* of this later.

How do the numbers in the selections table get changed? Magic? Wishful thinking? Clean living? Not quite. How that happens is controlled by other aspects of your database design. But to give you an idea of one solution to that 'problem' I'll show you what the Customer Acquisition Selection Handler (CASH) system could look like when the customer is using it:



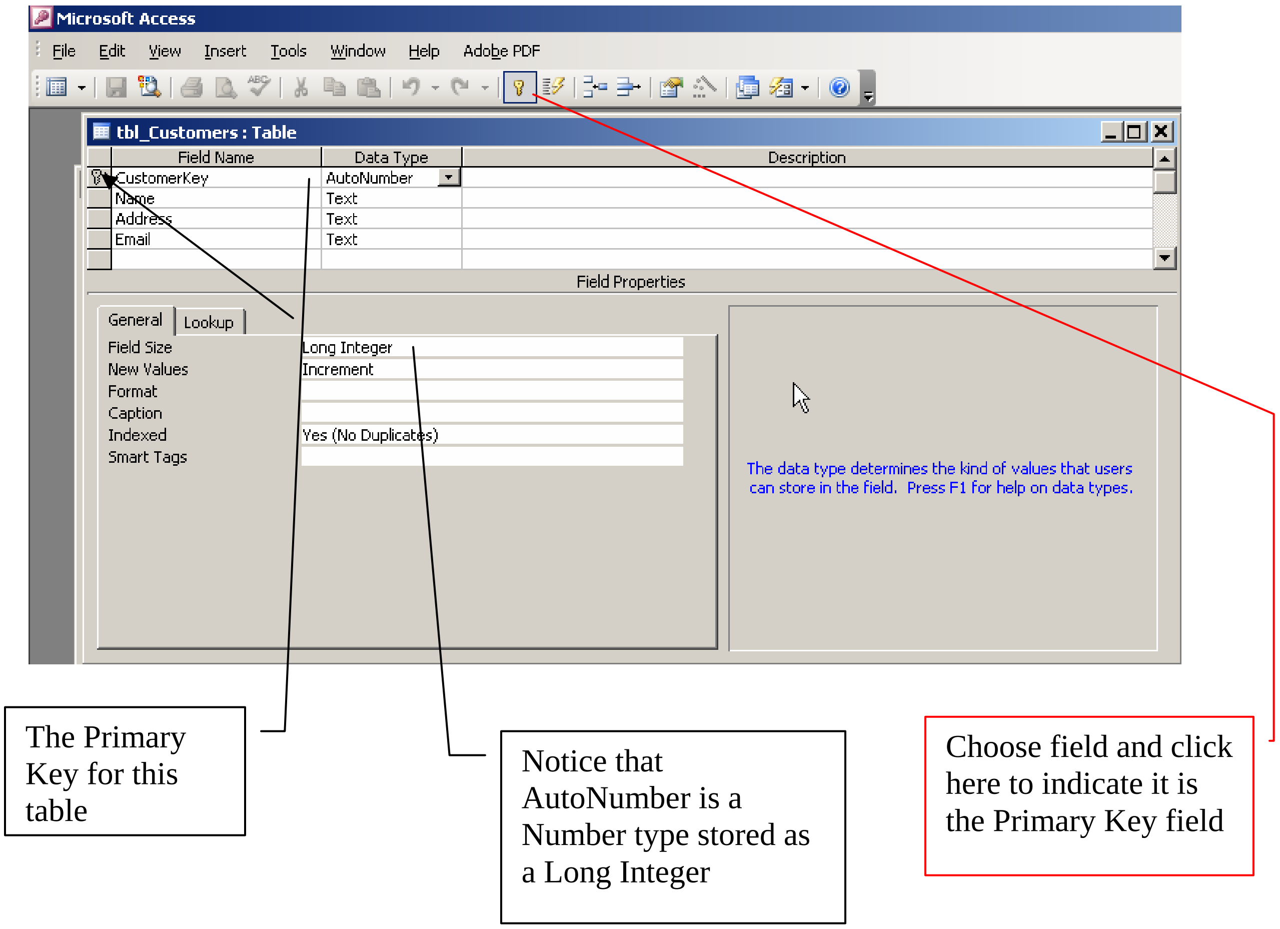
CustomerKey	1	Name	Elmo	Address	Somewhere	Email	noreply@

CustomerKey:	Model	Color	Engine	Transmission	Interior
1	4 Door	Black	4 Banger	Automatic	Vinyl
1	2 Door	Red	6 Pack	Manual	Leather
1	2 Door	Black	4 Banger	Automatic	Cloth

2 Door
4 Door

You'll notice that the Model field turns into a list when you choose that field. The same thing happens for each of the other fields in the frmSelectionsSubform. The values you can choose are controlled from the contents of 5 lookup tables, one for each information group. Another thing to note is the "CustomerKey" field that exists in both the main form, frm_Customers, and the subform. That value is the Primary Key in tbl_Customers and is the foreign key in tbl_SelectionsMade. That is how we know which selections were made by which customer.

How did I get those things defined? Here we get into the mechanics of Access itself. During my table definition I indicated that the CustomerKey field would be type “AutoNumber” and that I wanted Access to make sure that the same number was never used twice in that table.


Microsoft Access
File Edit View Insert Tools Window Help Adobe PDF
tbl_Customers : Table
Field Properties
General
Lookup
The Primary Key for this table
Notice that AutoNumber is a Number type stored as a Long Integer
Choose field and click here to indicate it is the Primary Key field

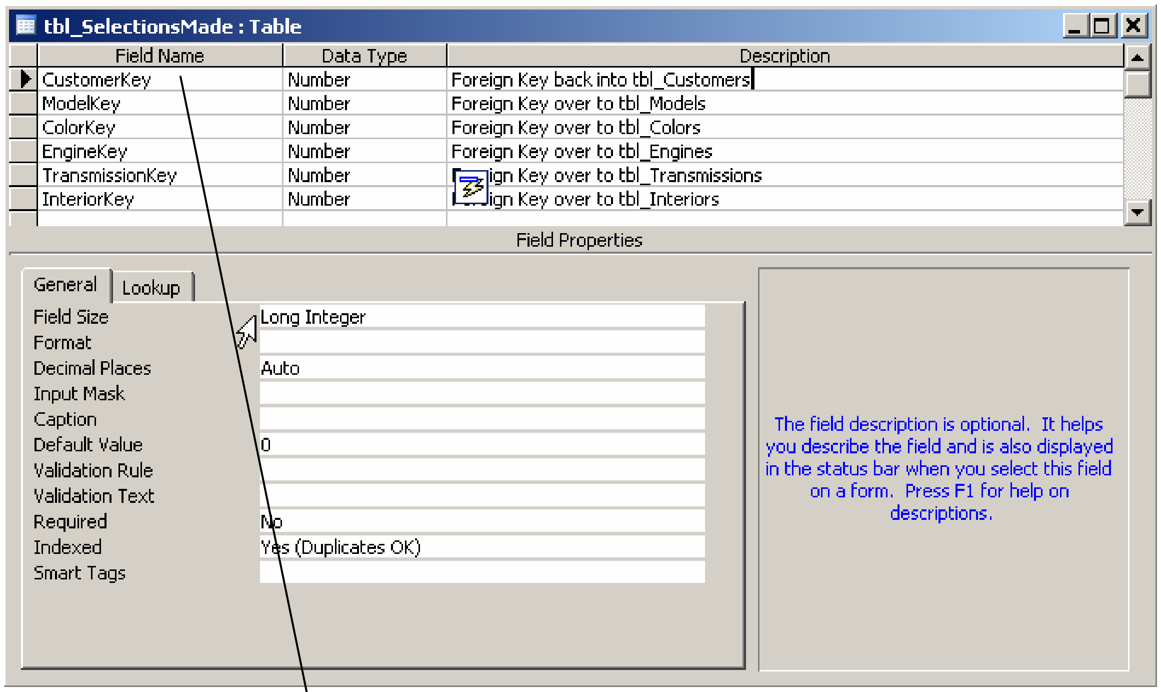
Field Name	Data Type	Description
CustomerKey	AutoNumber	
Name	Text	
Address	Text	
Email	Text	

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)
Smart Tags	

The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

Initially I just set CustomerKey to type AutoNumber, which as you’ll notice is a special case of Number | Long Integer typing. It is special in that the values are created automatically by Access when new records are created. But knowing that it is a Long Integer gives you a big clue on how to format the associated Foreign Key in related tables. By choosing a field and then clicking the Primary Key icon in the toolbar, the field is automatically set up to be the key field (an entry is required and it must be unique within the table and it is Indexed for fast lookups).

On the next page we will look at the related table, tbl_SelectionsMade is set up initially.


tbl_SelectionsMade : Table

Field Name	Data Type	Description
CustomerKey	Number	Foreign Key back into tbl_Customers
ModelKey	Number	Foreign Key over to tbl_Models
ColorKey	Number	Foreign Key over to tbl_Colors
EngineKey	Number	Foreign Key over to tbl_Engines
TransmissionKey	Number	Foreign Key over to tbl_Transmissions
InteriorKey	Number	Foreign Key over to tbl_Interiors

Field Properties

General | Lookup

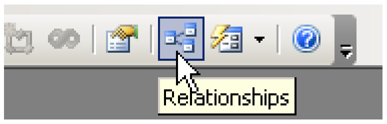
Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	Yes (Duplicates OK)
Smart Tags	

The field description is optional. It helps you describe the field and is also displayed in the status bar when you select this field on a form. Press F1 for help on descriptions.

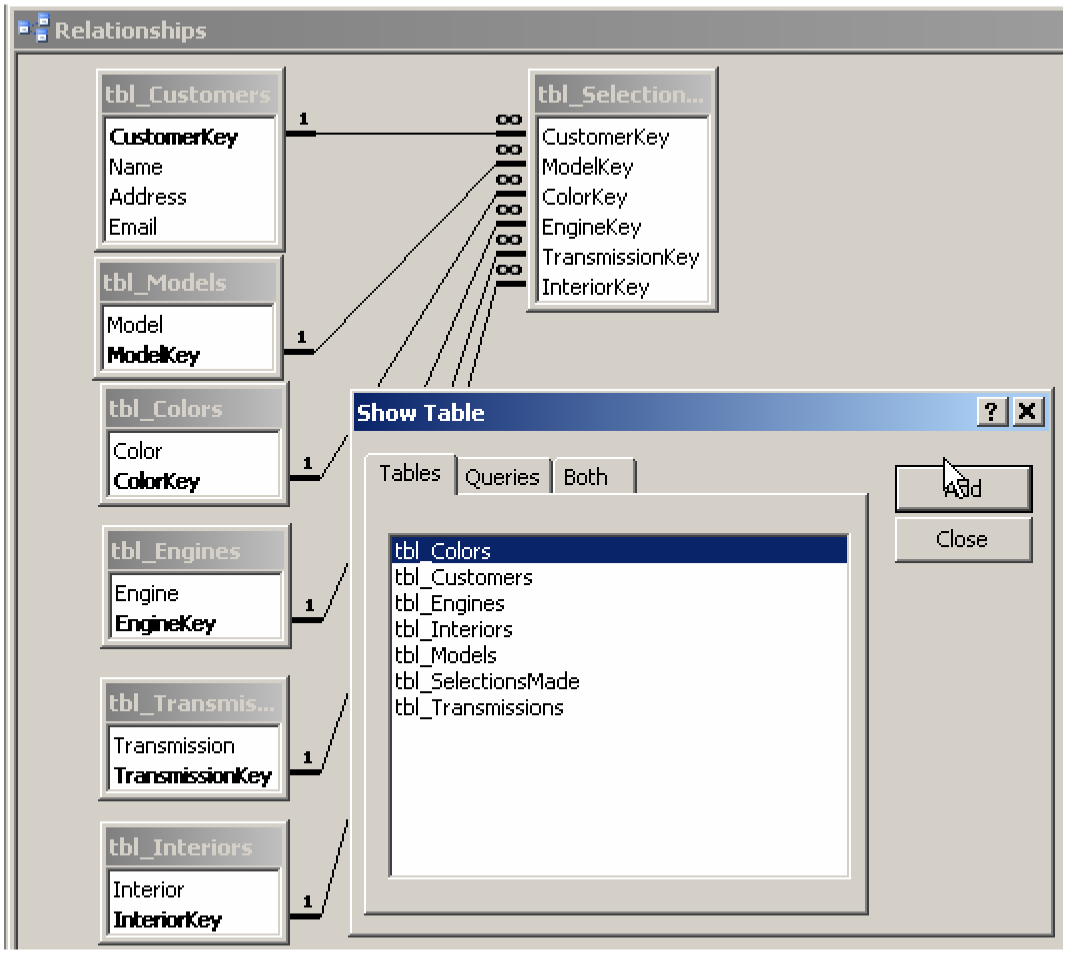
I gave mine same name as in the primary table. Notice that I set it up as Number with sub type of Long Integer. This is so that copies of the Primary Key from tbl_Customers will fit into it properly.

You may also notice that I have chosen to just store the Primary Key values from my lookup tables in this table. While this makes interpreting information by looking at the raw table data difficult, it presents no problems when you set up queries to tell you what it all means.

You now ask yourself "how does Access know that the Primary Key over in tbl_Customers is the Foreign Key in tbl_SelectionsMade?" and same question regarding all of the other entries in the table.

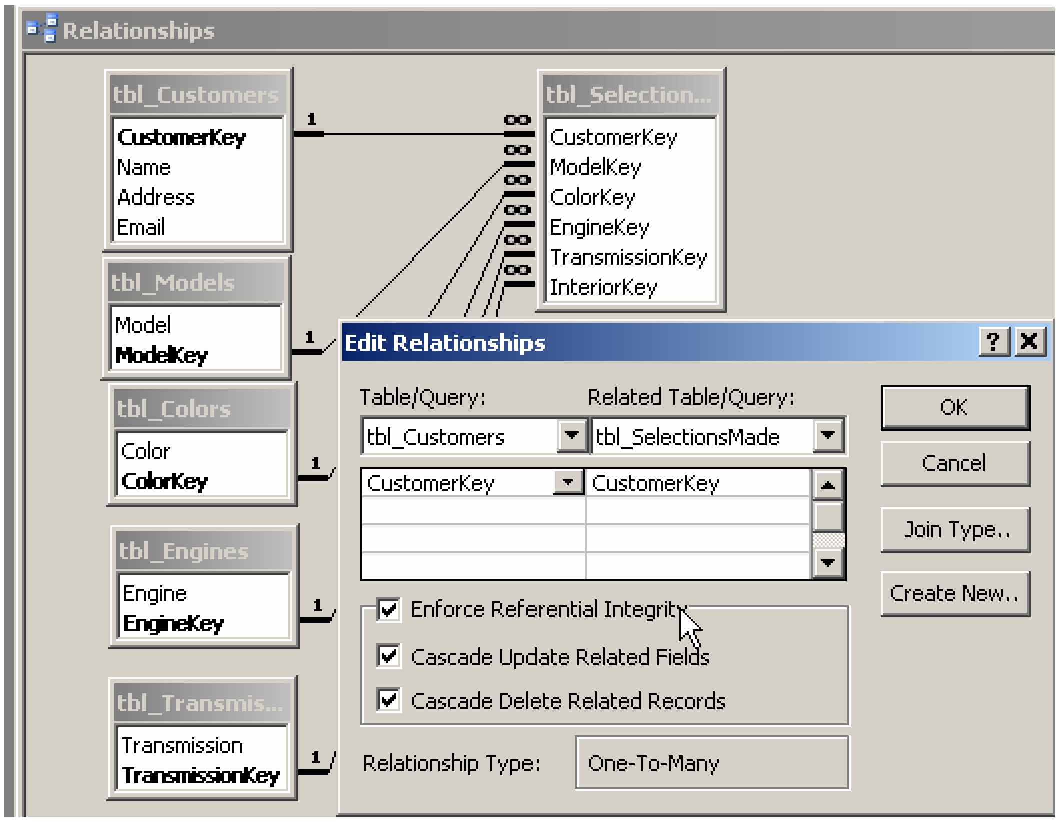


The Relationships window opens up when you choose the Relationships icon from the toolbar.



Whoa! Scary! Not really. Initially the Relationships window shows up completely empty and you have to use Right-Click | Show tables and that brings up the Show Table window. You just choose the tables and/or queries you want to set up relationships between.

Then for each primary or lookup table I simply clicked-and-dragged the field in the primary table over to the field name in the related table that I wanted to be related to it and dropped it there. Other options pop up when you do that:



Table/Query: tbl_Customers — Related Table/Query: tbl_SelectionsMade

Table/Query	Related Table/Query
CustomerKey	CustomerKey

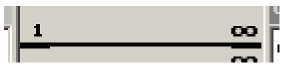
- [✓] Enforce Referential Integrity
- [✓] Cascade Update Related Fields
- [✓] Cascade Delete Related Records

Relationship Type: One-To-Many

The big one is the "Enforce Referential Integrity" – choosing that option says that before a value can exist in the related table, it must already exist in the primary table. After that you can choose to "cascade" update/delete related records. I chose to use both of those options in this case.

Cascade Update Related Fields – mostly you want this one checked if you've chosen a field that is a primary key somewhere that a user could change – such as SSAN (as when you find out you typed one in error). If the primary key value is changed, then that change will be made automatically where ever it is used as a foreign key.

Cascade Delete Related Records – be careful here. In this case, if we delete a Customer record, we want to get rid of all related selections records. But if for some reason we wanted to keep those related records for analysis or historical reasons, I wouldn't have chosen that option. If the option is *not* checked, then you won't be able to delete Customer records until you somehow otherwise delete the related Selections records.



What the heck is that? That is Access's way of telling you that this is a one-to-many relationship. That for each unique record on the "one" side, there can be:

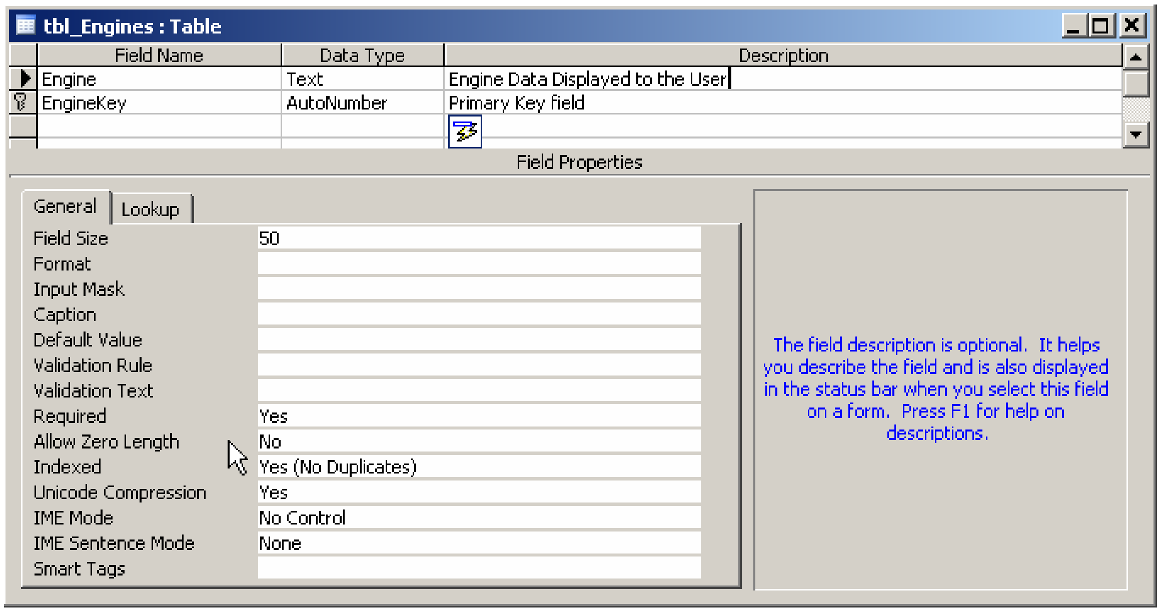
None,

One,

Or a zillion (depending on how much memory your computer has)

Related records in the "many" side table. There are other combinations as 1-to-1 and many-to-one and even many-to-many (esthetically displeasing and logic challenging: i.e. ugly and confusing).

We should revisit the design of some of the tables for a moment. Let's take one of the lookup tables, tbl_Engines, for example. The design of them all is pretty much the same, they have 2 fields – one to hold the information and a Primary Key field that other tables will link to them with.



tbl_Engines : Table

Field Name	Data Type	Description
Engine	Text	Engine Data Displayed to the User
EngineKey	AutoNumber	Primary Key field

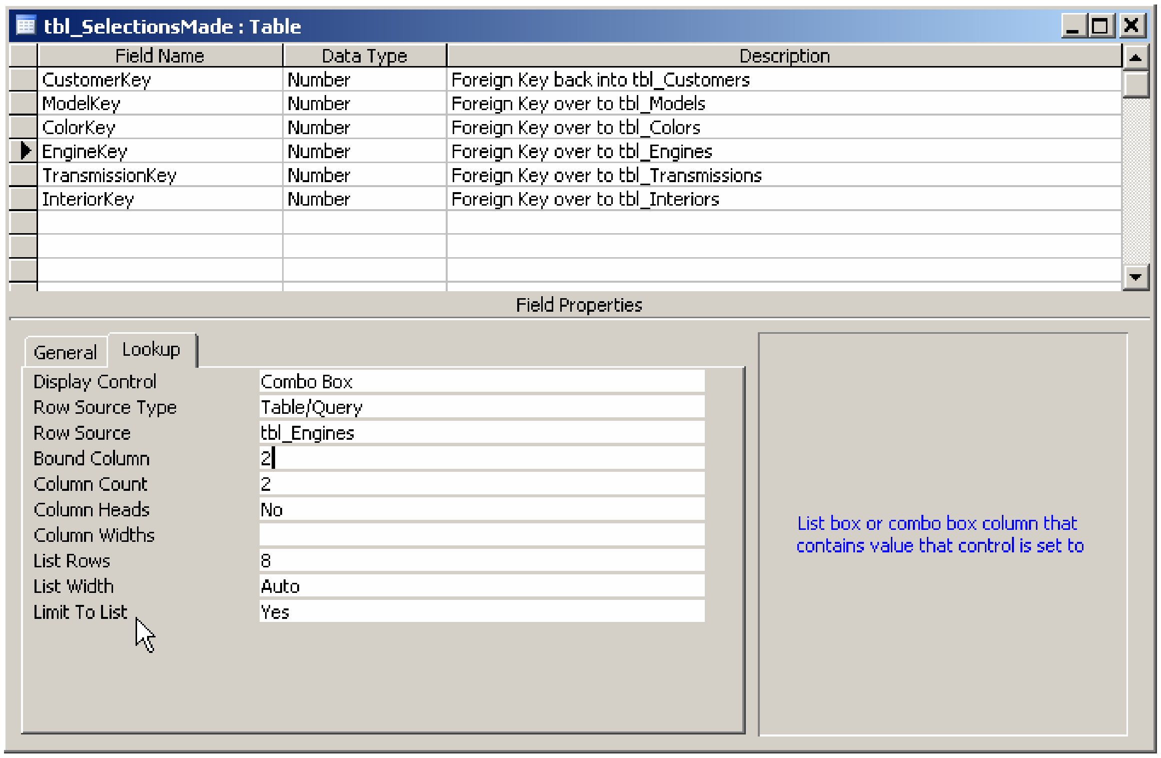
Field Properties

Field Size	50
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

The field description is optional. It helps you describe the field and is also displayed in the status bar when you select this field on a form. Press F1 for help on descriptions.

I intentionally set the primary key field up as the second field in this table. You will see why later. Notice the definition (properties) for the data field [Engine]. It is required information and you cannot just put in a nothing (no zero length data entries). It is indexed just in case we ever want to look using the word descriptions rather than the numeric value of the primary key.

Keeping this in mind, lets now look at how we define our tbl_SelectionsMade using the structure of these lookup tables to our advantage.



tbl_SelectionsMade : Table

Field Name	Data Type	Description
CustomerKey	Number	Foreign Key back into tbl_Customers
ModelKey	Number	Foreign Key over to tbl_Models
ColorKey	Number	Foreign Key over to tbl_Colors
EngineKey	Number	Foreign Key over to tbl_Engines
TransmissionKey	Number	Foreign Key over to tbl_Transmissions
InteriorKey	Number	Foreign Key over to tbl_Interiors

Field Properties

General | Lookup

Display Control	Combo Box
Row Source Type	Table/Query
Row Source	tbl_Engines
Bound Column	2
Column Count	2
Column Heads	No
Column Widths	
List Rows	8
List Width	Auto
Limit To List	Yes

List box or combo box column that contains value that control is set to

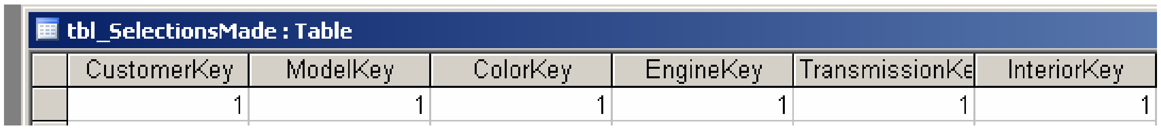
I chose the EngineKey field here in the table and then clicked on the Lookup tab of the Properties section. Initially it was rather sparse – there was no entry in the [Row Source] area and the [Bound Column] and [Column Count] values were both 1, and the [Limit to List] setting was set to No.

First thing I did was choose the table to be the source of the lookup for this field. If you click in the [Row Source] entry area a pull-down list shows up allowing you to choose from tables and queries that have been defined. I chose tbl_Engines.

Then I told it that the "bound column" was the second column (second field) of the table, not the first, and went further to tell it that there were only 2 columns in the table. I could have lied, but didn't.

I set the [Limit to List] entry to Yes so that people cannot arbitrarily enter anything they want – they have to choose from the list or type an exact match to an entry in it.

Why not have the primary key field in those lookup tables be the first field? Because by having them as the 2nd field, we get Access to help us help ourselves. Remember where we looked at tbl_SelectionsMade back on page 8 where it looked like this:

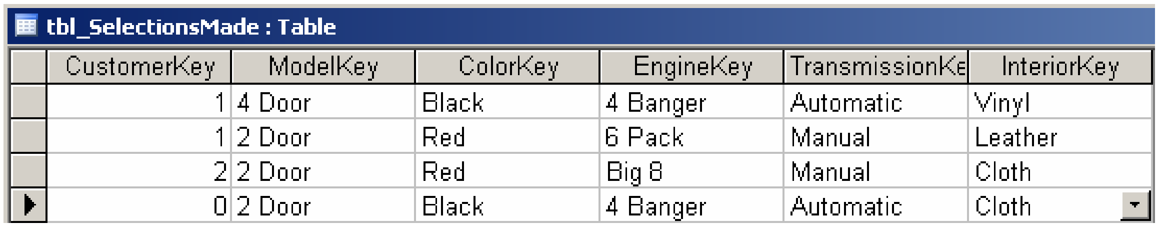


tbl_SelectionsMade : Table

CustomerKey	ModelKey	ColorKey	EngineKey	TransmissionKe	InteriorKey
1	1	1	1	1	1

Well, using this scheme changes the appearance of things in it:

Now entries in it look like this when viewed:

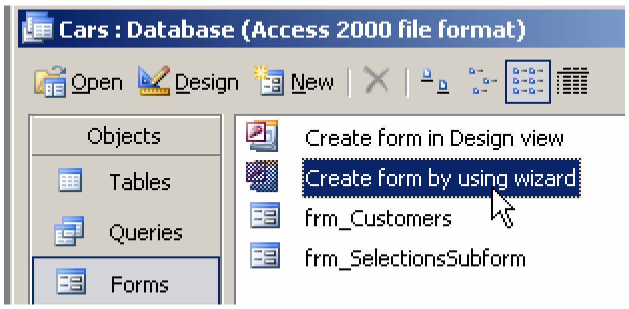


tbl_SelectionsMade : Table

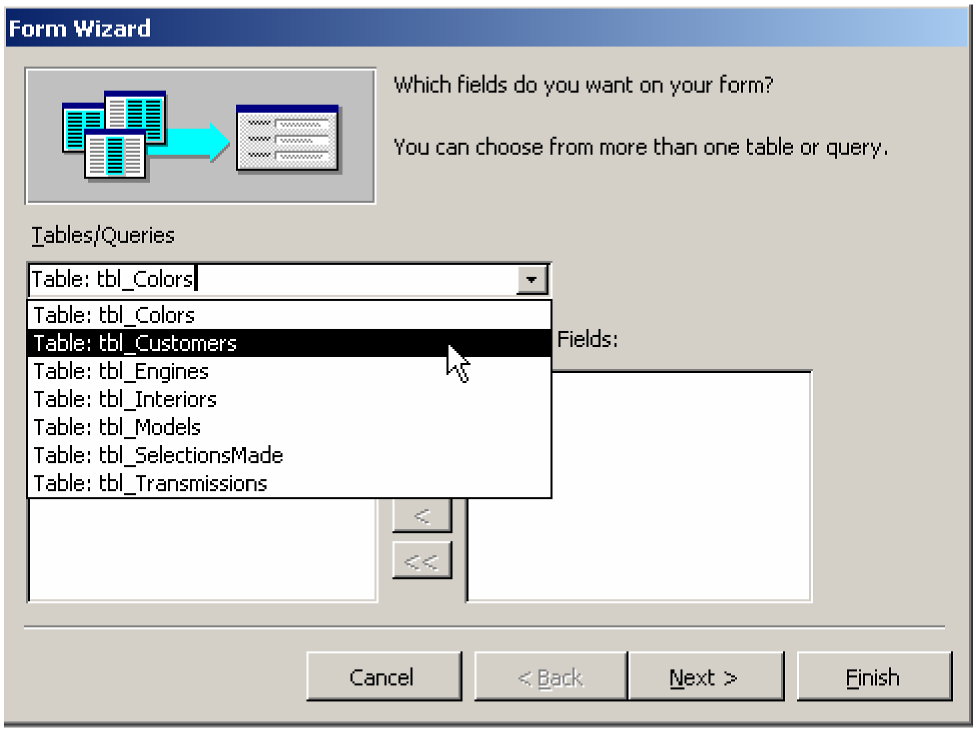
CustomerKey	ModelKey	ColorKey	EngineKey	TransmissionKe	InteriorKey
1	4 Door	Black	4 Banger	Automatic	Vinyl
1	2 Door	Red	6 Pack	Manual	Leather
2	2 Door	Red	Big 8	Manual	Cloth
0	2 Door	Black	4 Banger	Automatic	Cloth

Aha! Words instead of numbers for everything except the CustomerKey field. Plus all of the choices fields now have a built-in combo box for choosing from the lists of available options. We didn't change data types or anything in this table, in reality the table still is storing the Primary Key field value of each of the lookup tables, but it is displaying the contents of the first field (column) in the table (or query).

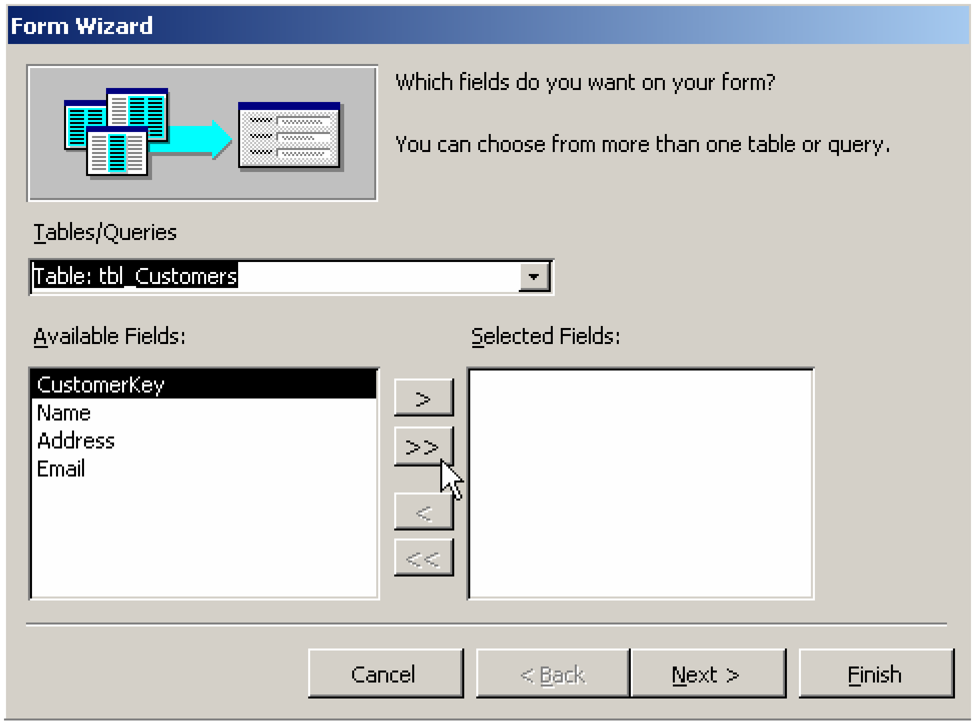
How can we take all that we've done so far and build up something to allow our customers to use it easily? That's where Access's AutoForm Wizard comes to our rescue.


Cars : Database (Access 2000 file format)
Open
Design
New
Objects
Tables
Queries
Forms
Create form in Design view
Create form by using wizard
frm_Customers
frm_SelectionsSubform

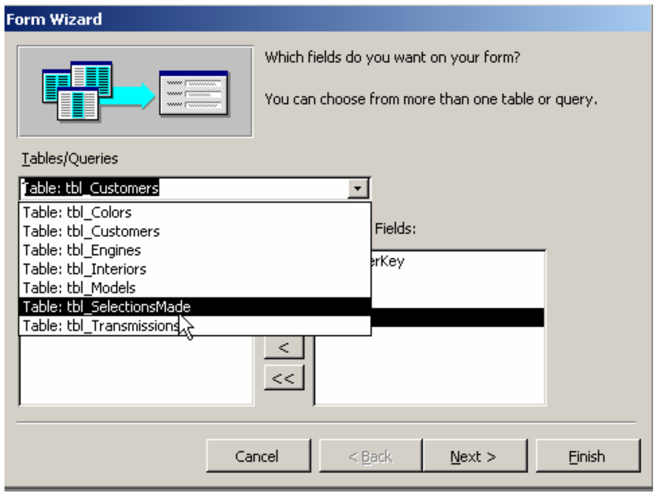
We choose [Forms] in the database window and then request Wizard help for creating a new form. Even though I've already created forms to show earlier, we will create some now to show how it was done – we'll just use different names for them.



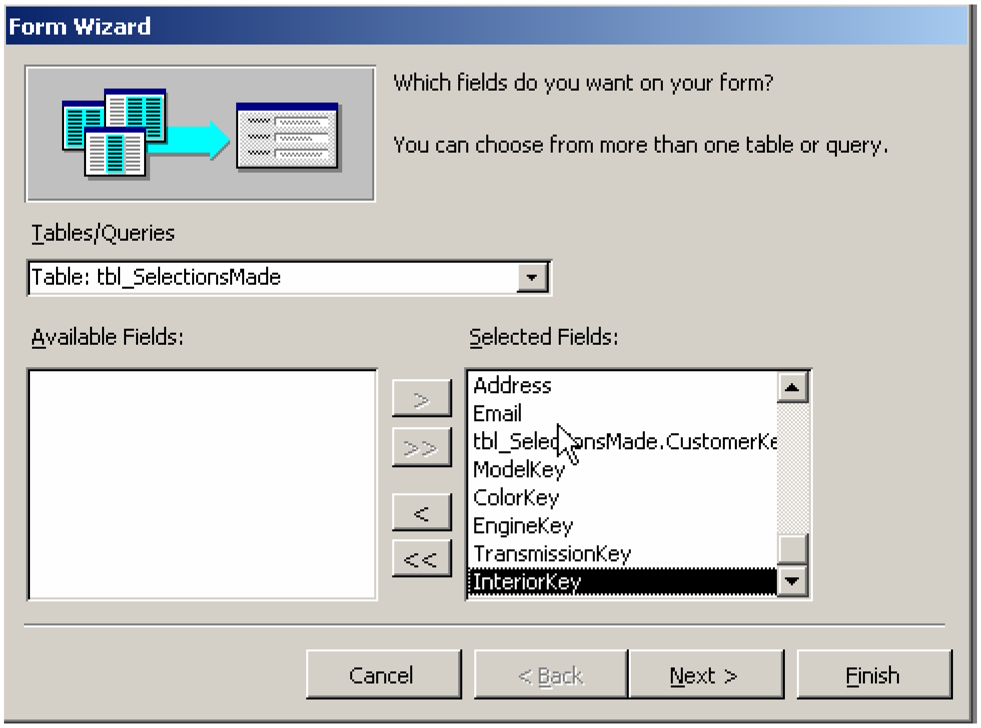
The first thing we do is choose the table that will be the "top level" source of information (the "one" side of our one to many relationship.



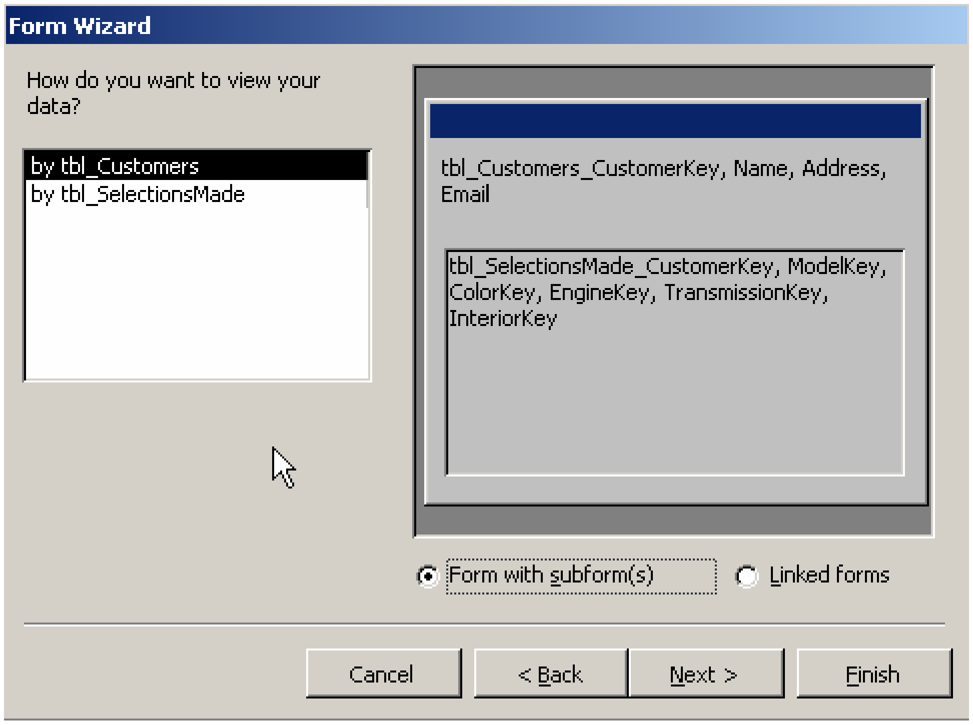
Once we've chosen the table, the fields in the table become available for choosing to become part of our new form. You can choose them one at a time and use the single [>] button to move them into the form, you can use the .[>>] button to move all remaining in the left window over into the right window. You may want to move some one at a time, because the sequence they end up in over in the right-hand window is the sequence they will appear in on your form. I just clicked on the [>>] button.



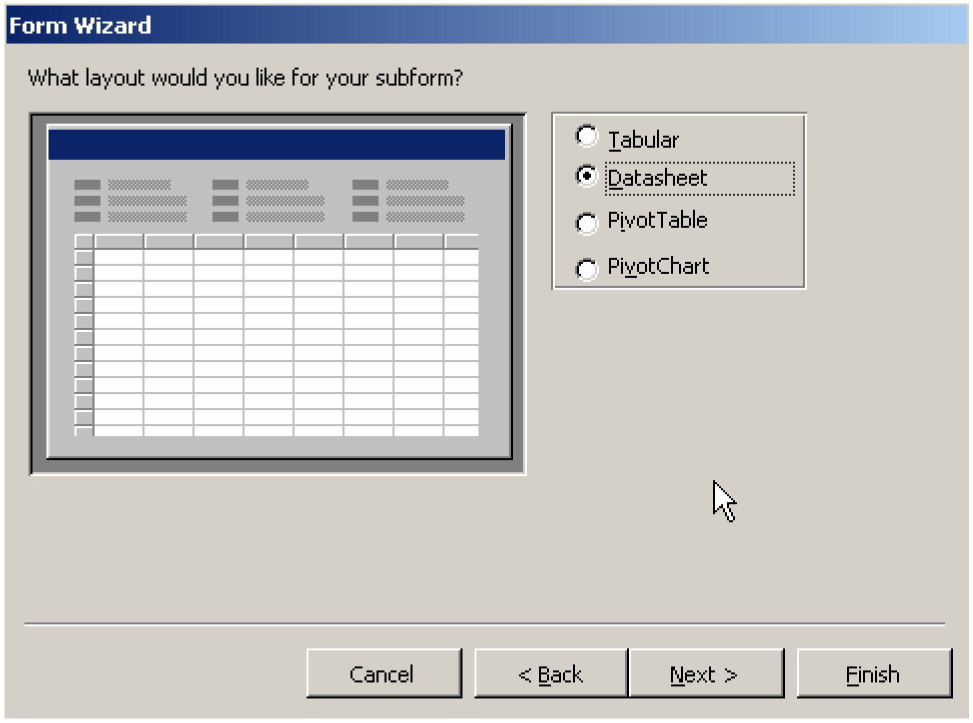
At this point you could click [Next] and move on to the next stage, ***but*** we want to create a related form that will automatically use the Primary Key from this table to become part of the records controlled in what is known as a "subform". You can actually go to 3 levels of forms this way. To do this, after moving the fields from tbl_Customers into the right hand window, I chose the next table, tbl_SelectionsMade and will once again move all of its fields into the right-hand window.



Now that window looks like this and we can click the [Next] button to move on now.


Form Wizard
How do you want to view your data?
by tbl_Customers
by tbl_SelectionsMade
tbl_Customers_CustomerKey, Name, Address, Email
tbl_SelectionsMade_CustomerKey, ModelKey, ColorKey, EngineKey, TransmissionKey, InteriorKey
Form with subform(s)
Linked forms
Cancel
< Back
Next >
Finish

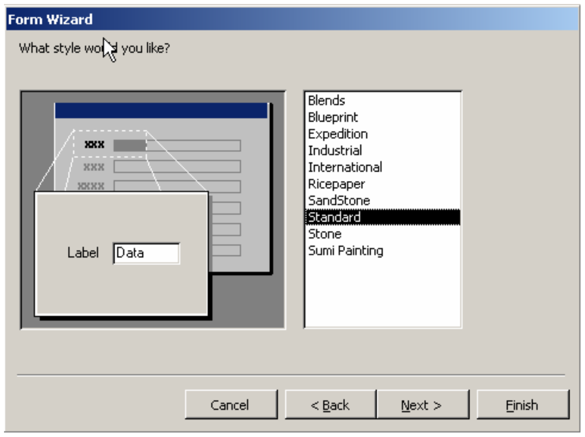
Which is pretty much what we want. There is one choice you may want to look at and possibly prefer in some cases: the [Linked forms] option. Instead of a form within a form, you get the main form and a toggle button that displays/hides a separate form containing the related records. This might even be a good place to use that option, but we will pass on it for now. Clicking [Next] brings this up:


Form Wizard

What layout would you like for your subform?

Tabular [✓ Datasheet] PivotTable PivotChart

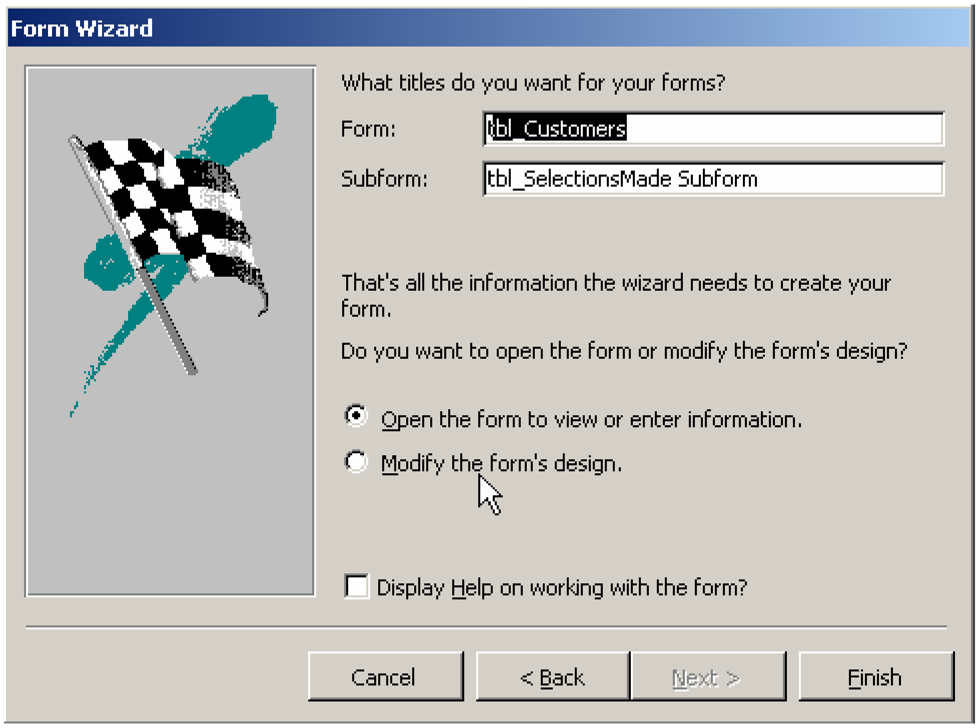
This is just asking us how we want that subform to look. The two most obvious choices are the top two, and the more compact is the default ‘Datasheet’ view. Again, click [Next]


Form Wizard

What style would you like?

Blends
Blueprint
Expedition
Industrial
International
Ricepaper
SandStone
[✓ Standard]
Stone
Sumi Painting

This one is simply checking with us as to how pretty we want the form to be. You can include pictures as backgrounds on a form and set the appearance of controls and things pretty much as you like to make them, prettier, emphasized, or just ‘the way you like them’. Click [Next] to continue to the next step.



What titles do you want for your forms?

Form: tbl_Customers

Subform: tbl_SelectionsMade Subform

That's all the information the wizard needs to create your form.

Do you want to open the form or modify the form's design?

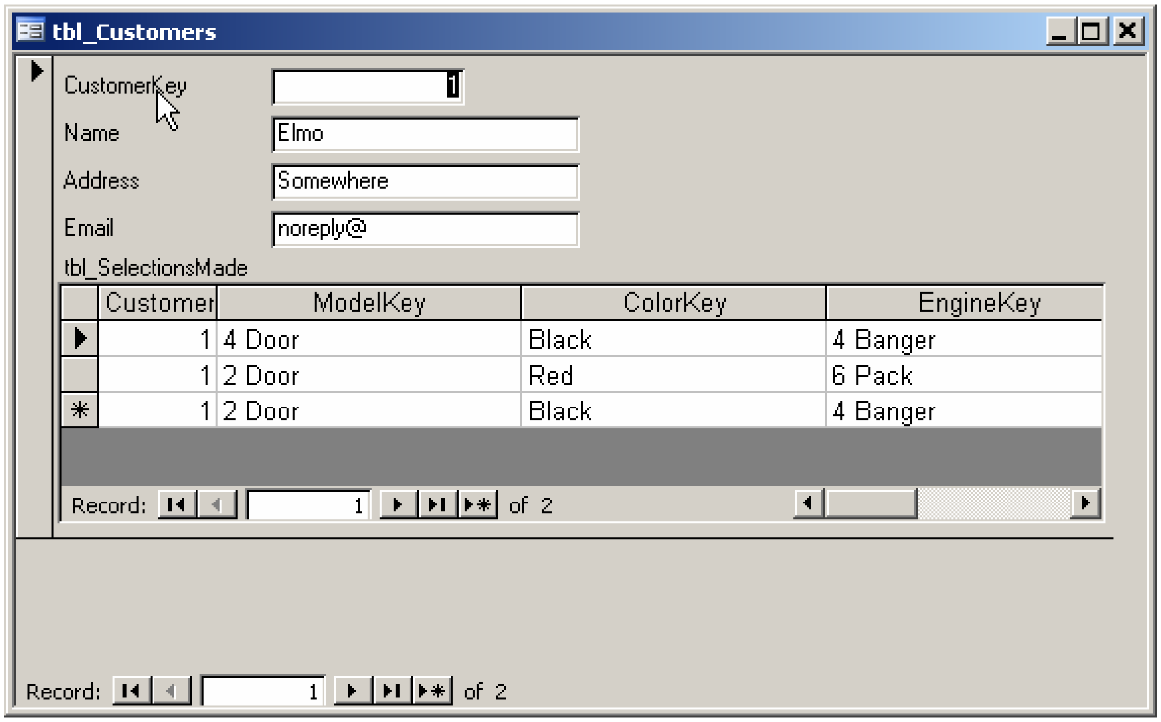
[✓ Open the form to view or enter information.]

Modify the form's design.

Display Help on working with the form?

Notice that this is asking for the titles to appear on the form ***not*** the actual name of the forms as they will be shown in the database window or referenced in code, queries or elsewhere. You can change these to suit your frame of mind. Click [Finish] and you are done unless you want to modify the design or layout some.

The newly created forms look like this:



CustomerKey	1
Name	Elmo
Address	Somewhere
Email	noreply@

tbl_SelectionsMade

Customer	ModelKey	ColorKey	EngineKey
1	4 Door	Black	4 Banger
1	2 Door	Red	6 Pack
1	2 Door	Black	4 Banger

Record: 1 of 2

Record: 1 of 2

It is kind of obvious that you need to make some modifications. For one thing, you can hide the CustomerKey on both forms. There is no reason for anyone to ever even be aware of its existence at this level of use. You could re-arrange the information in the main form to be more compact, and even change labels in the sub-form. Finally, you might resize some things so you can see all of the option choices across the sub-form without having to scroll horizontally. But it is usable at this point in time.

References:

Mike Hillyer - Mike's Bookshop MySQL related

http://dev.mysql.com/tech-resources/articles/intro-to-normalization.html

Databases: Normalizing Access Data

Beth Melton, MVP, MOS Master Instructor

http://pubs.logicalexpressions.com/Pub0009/LPMArticle.asp?ID=88

Microsoft on normalizing:

http://support.microsoft.com/kb/283878/EN-US/

George Hernandez

http://www.georgehernandez.com/h/xDatabases/aaIntro/DBStructure.htm