Tutorial for Begginers: How To Create Your Access Tables

By Greg Krajenta (Kraj)
Contributors:
WindSailor
Len Boorman

Introduction

1. Understanding Your Needs
2. The Thought Process: Relational Databases and You
3. Normalization
    a. 1st Normal Form
    b. 2nd Normal Form
    c. 3rd Normal Form
    d. Other Normal Forms
4. Table Definition Tips
    a. Primary Key
    b. Field Attributes
    c. Naming Conventions

Introduction

Welcome to the Beginner's Guide to Creating Access Tables. This is intended for anyone who is just starting to work with Access, whether you're a student, a self-taught dabbler, or a professional who's just been handed a database assignment regardless of the fact you don't work with databases.

The purpose of this guide is to communicate the fundamental concepts you need in order to build a strong foundation for your database so you don't tear your hair out later. And the intention is communicate in plain English, so you don't have to already be an expert to understand what people are talking about. This is a living document, intended to improve and grow with time. Please leave any suggestions, additions and feedback in the thread. If a section is confusing or feels incomplete, tell me. I do, however, want to limit the scope of this so it does not become overwhelming. I'd rather see major topics get their own tutorial.

From time to time I'll expand on a topic via a *Note. I provide this information because I think it is useful, but set it aside because it is not essential and should not be focused on if it is confusing. Feel free to skip them entirely. Now, then… let's get started.

1. Understanding Your Needs

The first step in creating a database is to know what you need your database to do. This is the most important step but it is also a broader topic than I want to or can cover here thoroughly. In brief, you should have a clear idea of what problem you are trying to solve

or what role the database will have in your activity. If you are making the database for another person(s), be sure you know what it is they want and how they want to do it. Understanding how your database will be used is key to organizing it in a way that makes sense. Don't be afraid to sit down with someone who will be using your database and ask them to walk you through what they want. The better you understand what you're being asked to do, the better chance you'll create a database that's exactly what they want. If you need help in this area, get a hold of some introductory material on Systems Analysis and Design. A good place to start would be understanding 1-to-1, 1-to-Many and Many-to-Many relationships, as well as entities and attributes. You should also have a reasonable idea of what Access is capable of doing (*If you are unfamiliar with the basics of Access' capabilities, you should start with basic tutorials or obtain a beginner's guide of some sort). Once you have these factors in mind, you are ready to grab a pad of paper and a pencil.

*Note: Most texts on Systems Analysis and Design strongly advocate using data modeling conventions. In my experience, mapping your entities, attributes and relationships can be useful, especially if your database is complex. Mine tend to be small and simple, so I rarely need to do more than define my tables. My college classes pushed data flow diagrams as a means to model a database. Attention students: data flow diagrams are great if you are writing lots of code, but if your database does not include lots of code-based processing they are utterly worthless. Eschew them whenever possible. You hear it here first.

2. The Thought Process: Relational Databases and You

To effectively design a table in Access, you have to know how the tables work or you'll get nowhere. We'll start with a definition of "relational database":

> *"A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables."*

Helpful, right? Simply put, a relational database is a bunch of tables with information in them. Sometimes the same information is stored in more than one table, and those tables are related to each other because of that common information.

An example – Let's say you create a database that stores a name, home address, and car(s) owned. You make two tables, one for the address and one for the cars. They might look something like this:

Table 1

Name	Address	City	State	Zip
Eric	1234 Elm St.	Chicago	IL	60601
Stan	5678 Oak St.	Chicago	IL	60602

Table 2

Name	CarModel	Mileage
Eric	Testarosa	20,000
Eric	S-575	10,000
Stan	S-575	5,000

Since “Eric” appears in both tables, the database will match “Testarosa” and “S-575” with “1234 Elm, etc.” based on the common element, “Eric” and vice versa.

Additionally, you may have a table that stores information about cars. It might look like this:

Table 3

CarModel	Make
Testarosa	Ferrari
S-575	Lotus
Accord	Honda

In this example, Table 2 is used to join the information in Table 3 with Table 1. You now know that there is a Ferrari with 20,000 miles on it parked at 1234 Elm St. in Chicago because you know Eric lives at 1234 Elm St. (Table 1) and he has a Testarosa with 20,000 miles on it (Table 2), which is made by Ferrari (Table 3). The connections between the tables are known as “relationships”.

3. Normalization

You may be asking yourself at this time, “Why can’t you just put Ferrari and Lotus in the same table with the Eric?” At first glance it seems like a good idea that makes things a whole lot simpler and, when you’re dealing with very small sets of data and very simple databases, it is. (If this is the case, though, then an Excel spreadsheet might be a better option for you.) However, when you start working with large amounts of information, complex data or queries, many problems arise if you do not follow a certain standard for efficient and stable table design. This standard is called “normalization” or “normal form”. There are many online resources available that discuss normalization in great detail, such as Wikipedia, so I will not do that. I will instead provide the essential basics.

3.a. 1st Normal Form

A database is said to be in 1st Normal Form when no records have repeating data groups. This means two things: each field should contain one unique value (ie., a ‘CarModel’ field should not contain the data “Ferrari, Lotus”);

and each record should not have more than one field to store the same kind of data (ie., the records in Table 1 should not have fields such as 'CarModel1', 'CarModel2', etc.).

There are two reasons to eliminate repeating data groups. First, if you wish to search your database for a certain data, it is *much* easier to do so when the data is in a field by itself. Addresses are a prime example of this. If your address field says, "1234 Elm Street, Chicago, IL, 60601, USA), a simple query for everyone in Chicago will not find this record.

The second reason for eliminating repeating data groups is efficient use of space. Continuing with the address example, say we want to store both a home and business address for each person. If we put both addresses in the same record, we might have the following fields: Number, Street, St/Ave/Blvd/etc, City, State, ZIP, Country ***and*** BusinessNumber, BusinessStreet, BusinessSt/Ave/Blvd/etc, BusinessCity, BusinessState, BusinessZIP, BusinessCountry. If every entry in your database utilizes all those fields then you're fine, but if some entries only have one address your wasted space will add up very quickly. See section 4.b. for more on this.

To solve repeating data groups, make a new table with the repeating fields and an appropriate field from the first table (usually the primary key).

3.b. 2nd Normal Form

A database is said to be in 2nd Normal Form if it is already in 1st Normal Form and all the fields in a record are fully dependant on the primary key. This applies only to tables with a composite (also sometimes called a concatenated) primary key. (*See section 4.a. for more on primary keys.)

In the above example, Table 2 has a composite key; both the "Name" and "Car" fields combine to make a single key that identifies a unique record. Any other fields in a table with a composite key should be related to all the fields that make up the key and not just some of them. The "Mileage" field belongs in Table 2 because Mileage is dependent on the combination of model and owner, ie., only Eric's Lotus has 10,000 miles on it – not Eric, not all Lotuses. On the other hand, if we put the "Make" field from Table 3 into Table 2 it would violate 2nd Normal Form because all Testarosas are made by Ferrari; it doesn't matter who owns it. Therefore, "Make" is dependent on "CarModel" but not on "Name", so "Make" should not be in a table where "Name" is part of the key.

There are several reasons for eliminating partial dependencies. Primarily, they cause data to be repeated and stored more than is necessary. Adding "Make" to Table 2 means every time you enter a Testarosa, you must also indicate it is a Ferrari. It would be much easier and efficient to reference

another table that says any Testarosa is a Ferrari. Partial dependencies also make your data more vulnerable. If you have to enter Ferrari every time you enter Testarosa, you run an increased risk of making an error and indicating the wrong make for the Testarosa. Finally, if the data changes in one place it must be changed everywhere. If, for example, Ferrari decided to sell the brand name and design of Testarosa to Honda, you would have to go through your records and change every instance of Ferrari to Honda wherever it appears with Testarosa. If you put "Make" into its own table, as is the case in Table 3 as I've shown it, you would now only have to change Ferrari to Honda in one place. This makes updating your database easier and less vulnerable to mistakes. Admittedly, the example of changing the make of a car is far-fetched, but what if instead of make and model you have customer and address? Now you have a field that will frequently need to be changed.

To solve dependency problems, make a new table with the partially dependant field in it and the part of the key from the previous table on which that field is dependant.

3.c. 3rd Normal Form

A database is said to be in 3rd Normal Form if it is already in 2nd Normal Form and all the fields in a record are *only* dependant on the primary key. In the car example, let's say I added the field "Country" to Table 3 so it looked like this:

Table 3

CarModel	Make	Country
Testarosa	Ferrari	Italy
S-575	Lotus	Britain
Accord	Honda	Japan

This setup violates 3rd Normal Form because "Country" is dependant on "Make", not "CarModel" since the company based in Italy is Ferrari, not Testarosa, etc. Non-key dependencies have all the same negative impact on your database as the partial-key dependencies we eliminated in 2nd Normal Form.

To eliminate non-key dependencies, create a new table with the two fields that have the dependency you identified.

*Note: While the steps of normalization are referred to as rules, it is worth noting that they are not completely inflexible. Sometimes, for example, a partial dependency takes up a few bytes of wasted space per record but is such a small portion of the whole record that it is actually more efficient to keep it there instead of creating a new table. This is because the processing

time it takes for the database to access the extra table is greater than the processing time it takes to access the blank fields of the records. However, this is both a rare and controversial situation which you should not find yourself in unless you are the administrator of a very large and complicated database (in which case, if you are reading this you may be in over your head). So, I would not worry about this, just be aware of it.

The next point applies to a table at any stage of Normal Form, although it is most often included with 3rd Normal Form: your tables should not have any calculated fields. For example, if you have a record that stores a car dealer's order, it might include Dealership Name, Car Ordered, Quantity Ordered and Car Price (yes, including Car Price violates 1st Normal Form, but I'm trying to make a point here) and Total Price, which is the Car Price multiplied by the Quantity. The record actually should not include Total Price, because that amount can be calculated at the time you run your query. It is more efficient for a database to perform a calculation than it is to store data. In short, anything that can be calculated, summarized, etc., should not be stored as its own data in a field.

3.d. Other Normal Forms

1st, 2nd, and 3rd Normal Forms are all you are ever likely to need in order to have an efficient, stable database. However, databases that are very large or complex may benefit from 4th, 5th, 6th, or various other Normal Forms. I will not go into these in detail as many resources are available to explain them in greater detail and, honestly, I don't know much about them.

4. Table Definition Tips

Now that you have all of your tables written out in 3rd Normal Form, you're ready to build them in Access. Remember, at this point all your work so far should be on paper. I will assume you've taken my earlier advice and have some knowledge of Access database basics, or at the very least the instruction manual that can guide you through the steps of building the tables. Therefore, in these sections I will not go into great detail but instead provide some useful pointers.

4.a. Primary Key

Often, beginner literature will instruct you to set your primary key to whatever field uniquely identifies the record. So, in Table 1 the primary key would be "Name". This is not exactly wrong, but it is a bad idea. One reason is that data often changes, and when that data is a primary key it makes it much more difficult to make changes correctly. Also, if you delete a record (accidentally or purposefully) which contains a field that is used as a

primary key in another record, that other record either gets deleted as well or will cause errors. You will save yourself a lot of headache if every table uses an autonumber field as its primary key.

A term that is commonly used but is not apparently obvious what it means is "composite key" or "concatenated key". Simply put, a composite key is a primary key made up of more than one field. They are most often used in a table that joins two other tables that would otherwise not be related together, like Table 2.

4.b. Field Attributes

In a database, when you define a table you tell it what fields to have, what kind of data each field will store, and how much data each field will store. MS Access defaults to 50 characters of text. Whether or not you put data into this space, the record holds onto it. This means that, in the address example, if you do not utilize all the fields available you will be storing hundreds of characters worth of empty space in every record. This makes your database unnecessarily larger and slower. It is a good idea to adjust record sizes to fit your data (for example, in an address if you use only the two-letter abbreviation for a state, make the "State" field two characters). Also, if you are storing numbers that you intend on performing calculations on, you need to change the data type from Text to Number.

4.c. Naming Conventions

In brief, it is a good idea to give all your fields, tables, queries, etc., descriptive and consistent names. This will help you stay organized as you're designing as well as fix problems when you go back 6 months later and don't remember exactly what you did. It's up to you what to use, as long as it is consistent but there are generally-accepted guidelines available to reference.

4.d. Relationships

Once your tables are defined, you must define the relationships. You must do this before any data is in the tables. Relationships tell the database which fields are the same in each table. For example, "Name" appears in Table 1 and Table 2 but the database does not know the data is the same until you define a relationship. Defining relationships helps keep the database stable by preventing errors and makes designing forms and queries easier by automatically linking appropriate fields together. When you define a relationship, I suggest checking "Enforce referential integrity", which basically means "don't let me do anything that would screw up this data".