Data Normalization Fundamentals

by Luke Chung
President of FMS Inc.
http://www.fmsinc.com

Introduction

The ability to analyze data in Access is a fundamental skill that all developers must master. The better you are at organizing your data and knowing how to analyze it, the easier your application development will be. There are lots of ways to analyze data and different techniques must be used depending on your goal. However, there are a few fundamentals that must be understood:

- Data Normalization
- Separate Databases for the Application and Data

Data Normalization Overview

There are lots of articles and books on data normalization. They usually scare people including me. I am not going to get into a theoretical discussion of the pros and cons of data normalization levels. Basically, it comes down to this: how to store and retrieve data efficiently. This differs depending on the database used, so the more you understand how to manipulate data in Access, the more obvious the way you should store data in tables and fields.

A primary goal of good database design is to make sure your data can be easily maintained over time. Databases are great at managing more records. They are terrible if fields need to be added since all its queries, forms, reports, and code are field dependent.

Spreadsheet Gurus

Data normalization is a particularly difficult concept for spreadsheet experts. Having been a spreadsheet developer prior to using databases, I sympathize with those struggling to make the transition. The main reason you're using a database rather than a spreadsheet is probably because have so much data you can't manage it properly in Excel. The fundamental advantage of a database is that it allows your data to grow without causing other problems. The big disaster in most spreadsheets is the need to add new columns or worksheets (for new years, products, etc.) which cause massive rewrites of formulas and macros that are difficult to debug and test thoroughly. Been there. Designed properly, databases let your data grow over time without affecting your queries or reports. You need to understand how to structure your data so your database takes advantage of this. How you store your data is totally different from how you show it. So, stop creating fields for each month, quarter or year, and start storing dates as a field. You'll be glad you did it:

Federal Budget Non-Normalized : Table

ID	Data Type	1990	1991	1992	1993	1994	1995
1	Receipt	$1,031,309.00	$1,054,264.00	$1,091,300.00	$1,154,400.00	$1,258,600.00	$1,351,80
2	Outlay	$1,251,778.00	$1,323,011.00	$1,381,700.00	$1,409,400.00	$1,461,700.00	$1,515,70
3	Deficit	$220,469.00	$268,747.00	$290,400.00	$255,000.00	$203,100.00	$163,90
4	Human Resources	$619,327.00	$689,691.00	$772,440.00	$827,535.00	$869,414.00	$923,76
5	Defense	$299,331.00	$273,292.00	$298,350.00	$291,086.00	$281,642.00	$272,06
6	Other	$333,120.00	$360,028.00	$310,910.00	$290,779.00	$310,644.00	$319,86

Non-Normalized "Spreadsheet" Data

Federal Budget : Table

Year	Receipt	Outlay	Deficit	Human Resources	Defense	Other
1990	$1,031,309	$1,251,778	$220,469	$619,327	$299,331	$333,120
1991	$1,054,264	$1,323,011	$268,747	$689,691	$273,292	$360,028
1992	$1,091,300	$1,381,700	$290,400	$772,440	$298,350	$310,910
1993	$1,154,400	$1,409,400	$255,000	$827,535	$291,086	$290,779
1994	$1,258,600	$1,461,700	$203,100	$869,414	$281,642	$310,644
1995	$1,351,800	$1,515,700	$163,900	$923,765	$272,066	$319,869
1996	$1,453,100	$1,560,300	$107,200	$958,254	$265,748	$336,298
1997	$1,505,400	$1,631,000	$125,600	$1,019,395	$267,176	$344,429

Normalized Data

Both tables in the example above contain the same data, but they are distinctly different. Notice how the normalized table lets you easily add more records (years) without forcing a restructuring of the table. In the non-normalized table, adding next year's data would require adding a field. By avoiding the need to add a field when you get more data, you eliminate the need to update all the objects (queries, forms, reports, macros, and modules) that depend on the table. Basically, in databases, new records are "free" while new columns are "expensive". Try to structure your tables so you don't need to modify their fields over time.

Efficient Storage and Unique IDs

A fundamental principle of data normalization is the same data should not be stored in multiple places. Information that change over time such as customer names and addresses should be stored in one table and other tables referencing that information should link to it.

Unique IDs (key fields) with no connection to the data are used to link between tables. For instance, customer information should be stored in a customer table with a Customer ID field identifying the record. Access lets you use an AutoNumber field to automatically assign new ID numbers. It doesn't matter what the ID number is or whether they are consecutive. The ID number has no meaning other than identifying the record and letting records in other tables link to that record.

I've seen databases where people use ID numbers that combine a few letters of the last name, first name, and number. That makes no sense and creates a mess over time. The ID should just be a number and if you want your data sorted in a particular order, use a secondary index.

Data Normalization Extremes

It is important to not take data normalization to extremes in Access. Most people are familiar with separating Customers into its own table. If not, they quickly discover why they need to. But what about optional fields like telephone numbers: business phone, fax number, mobile phone, modem, home phone, home fax, etc.? Most customers won't have all those numbers, but you may want to store them for those that do. There are three approaches:

1. All the fields are in the Customer table.
2. A separate table is created for each type of telephone (a one-to-one link). The table would contain the customer ID and telephone number.
3. A telephone table is created with these fields: customer ID, the Telephone Type ID, and number so you could conceivably have unlimited phone numbers (a one-to-many link).

There are arguments for each alternative and to some extent it depends how well you know your data. Data normalization purists and programs such as Erwin often suggest separate table(s). Obviously option 3 is the most flexible since it allows you to support an unlimited number and type of phones. If you cannot limit the number of alternatives, this is your only choice. However, if you can limit the types, you should opt for option 1 in Access.

First, Access stores data in variable length records. One of the reasons for data normalization is to save disk space. Old file formats such as dBase, FoxPro, and Paradox stored data in fixed length records with each record taking the same space regardless if its fields were blank. The more fields, the larger the table. Not only is disk space is cheap today, Access records only grow if data is contained in them. Therefore, this kind of data normalization is not necessary for efficient data storage in Access.

Second, and more important, the retrieval of data across multiple tables may be an unnecessary hassle. If you are always going to show the customer's phone and fax number, retrieving those records out of another table is an unnecessary and will hurt your performance. The way Access is designed, it is much easier to just pick the fields from the Customer table rather than using a separate query or sub-report to grab each telephone type separately.

Storing Duplicate Data

It is very important to remember there are situations where you must store what seems like duplicate data. This is most often related to the passage of time and the need to