

Seven rules for a proper database design

By Ian Bicking and Brian Moloney

Databases are often the most important asset of a business. Within them are customer data, transaction histories, products, pricing, and employee information. How often have we heard the cry, "I wish system A would communicate with system B?" The incompatibility lies not with the system nor very often with the software – it rests at the feet of bad database design.

The structure of a database is crucial to its ability to transform raw data into usable information. Each database should conform to a set of standard rules designed to optimize its utility. These rules make a database a flexible, usable tool, and not just a place to store information.

These rules form the basis for any database design, from a spreadsheet of columns to a relational database with multiple tables.

Rule One: One piece of information per column

This rule addresses this kind of situation:

Name	Phone
Bob Dale	773-555-0534, 312-555-3456

By putting two phone numbers in a single field, it is difficult to perform searches or sorts. For example, if you want a report on all phone numbers within the 312 area code, this database will not be able to provide it. Also, if you have an auto-dialer using the database, it will dial a 20-digit number, which most certainly won't be what you want.

Rule Two: No table should have two columns with equivalent meanings

If you don't put multiple phone numbers into a single field, the next step is to simply make more fields:

Name	Phone1	Phone2
Bob Dale	773-555-0534	312-555-3456

This looks better and it is better, but there are still problems. Phone1 and Phone2 don't tell you very much about the numbers. Phone1 is probably preferred, but is it a daytime or nighttime number? What about people with more than two phone numbers? We can change the table to make it a little more flexible:

Name	Phone
Bob Dale	773-555-0534
Bob Dale	312-555-3456

This still doesn't answer some questions. Is one of these a cell phone, or home or work number? Rules 4 and 5 will show the conventional techniques for clarifying these questions.

Database design rules are also referred to database "normalization principals." A properly designed database is considered "normalized."

Rule Three: Strive for uniqueness

All is well and good, and then another Bob Dale comes along. Now what? How do we know which Bob is which? To avoid this, it is important to have unique identifiers. Identifying each with a unique code, like a Social Security Number, an email address, or some other unique identifier does this. This is referred to as a primary key. It also introduces us to the concept of tables, or mini-spreadsheets used to further break data down. Below is a table that assigns a unique ID to each of our Bobs. This unique ID is then used to associate the proper phone number with the proper Bob.

Person_ID	Name
1	Bob Dale
2	Bob Dale

Person_ID	Home_Phone
1	773-555-0534
1	312-555-3456

Rule Four: Plan for multiple values

Let's look at addresses now. Imagine we have a table with a row for every person, and a separate table for addresses.

Person Table		
Name	Business_Address_ID	Home_Address_ID
Bob Dale	1	2

Address Table					
ID	Street1	Street2	City	State	ZIP
1	123 Main Street	Suite 100	Heresville	IL	65432
2	10 Center Street		Theresville	NY	12345

It looks like a normalized structure, but what happens if Bob is a doctor and has three office addresses? What if you are shipping goods and you want to offer the ability for customers to ship to many addresses? You probably didn't think to include a Moms_Home_Address field.

The solution is to create another table that relates people to addresses, like:

Person_Address Table		
Person_ID	Address_ID	Type
1	1	Business
1	2	Home

Granted, we're pretty far from a simple Excel spreadsheet, often referred to as a flat-file database. The structures are getting more complicated as we move towards a relational database, but this is the only way a database can reliably handle an unlimited number of addresses for each person.

Rule Five: Don't repeat yourself

One of the basic principles of normalization is that any one piece of information should live only one place in the database. If it is located in multiple places there

Jargon Watch:

When each column is specific and there is only one piece of data in a cell, the resulting database is referred to as "atomic."

might be typos, your searches might not work and your update process may not be complete and consistent.

Going back to the address example in Rule 4, what do we do when more than one person lives in Theresville? When the same data is present in multiple locations within the database, it can appear in slightly different forms, causing many problems. For instance, if the city name is incorrectly entered as "Theres Ville" in one location, and "Theirsville" somewhere else, the computer can't tell those are the same cities as "Theresville" (computers are dumb like that). When we avoid repeating ourselves we avoid these issues. To do this, we break it down into the following:

Address Table			
ID	Street1	Street2	ZIP
1	123 Main St.	Suite 100	65432
2	10 Center St.		12345

ZIP_City_State Table		
ZIP	City	State
65432	Heresville	IL

This rule should be weighed carefully to avoid over-normalization. For example, what if two people live at the same address? You might be tempted to store the address only once and relate both people to it. What happens if one person moves? You might accidentally update the address for both. Sometimes repetition is necessary.

Rule Six: One column shouldn't be derivative of any other column in the same table

Each column should be an independent piece of data. If one column is derivative, that means you could determine its value from other values in the database. If you can do that, why include the value at all, why not just calculate it when you need it?

An example:

Person Table		
Name	First_Name	Last_Name
Bob Dale	Bob	Dale

You can probably tell that the Name field is a combination of the First_Name field plus the Last_Name field. So Name should not be included at all because what if it *isn't* always the first name plus the last name? You'll come to depend on that equivalency and later you may encounter inconsistent data that causes problems.

Also, it becomes unclear how to update values. If you change the First_Name to "Robert" does the Name field change as well? Must it be updated separately?

Rule Seven: Use appropriate types and constraints

When creating formal database designs, you typically tell the database what type of data it should expect in each column. Name is a string (just plain old text). Birthday is a date. Number-Of-Items-Purchased is an integer number.

This helps the database because it can treat the values much more intelligently when it knows what type they are. For instance, consider "8 Jun 1954" and "3 May 1973". If the database knows these are both dates it can tell you that "8 Jun 1954" occurred before "3 May 1973." But if it compared these two values as simply strings of characters, it would think just the opposite (since 3 comes before 8).

There are five levels of normalization, however most database designs function at the 3rd level, also known as 3NF, or third normal form.

Numbers can likewise become confused this way. The number 11 is larger than the number 5, but 11 "the string" is *less* than 5, because the computer compares the first characters 1 and 5, and finds 1 to be smaller (another example of dumb computers).

Types also mean validation. It means you can't put "30 Feb 2002" in a date field because that date doesn't exist. Likewise, you can't put "!2" into a number field.

Constraints take this validation further. They allow you to restrict values, like forcing a certain date to always be in the past (should be true for all birthdays), or a number to always be positive (like an order quantity).

The best time to correct data integrity problems is during input. The person entering the data is in the best position to resolve problems immediately. A year later when a report produces some odd results, it can be difficult to diagnose and correct the problem.

Conclusion

By adhering to these seven rules for proper database design, a business can maximize the performance of one of its most valuable assets. A properly designed database will have the flexibility to analyze data in new and exciting ways, the capacity to grow as data grows more abundant and complex, and the ability to protect investment by remaining transferable.

*Imaginary Landscape, LLC
5121 N. Ravenswood Ave.
Chicago, Illinois 60640
(773) 275-9144*

*<http://www.imagescape.com>
research@imagescape.com*