CHAPTER 8

Patterns and Anti-Patterns

I like rhyme because it is memorable; I like form because having to work to a pattern gives me original ideas.

—Anne Stevenson

There is an old saying that you shouldn’t try to reinvent the wheel, and honestly, in essence it is a very good saying. But with all such sayings, a modicum of common sense is required for its application. If everyone down through history took the saying literally, your car would have wheels made out of the trunk of a tree (which the Mythbusters proved you could do in their “Good Wood” episode), since that clearly could have been one of the first wheel-like machines that was used. If everyone down through history had said “that’s good enough,” driving to Wally World in the family truckster would be a far less comfortable experience.

Over time, however, the basic concept of a wheel has been intact, from rock wheel, to wagon wheel, to steel-belted radials, and even a wheel of cheddar. Each of these is round, able to move itself, and other stuff, by rolling from place A to place B. Each solution follows that common pattern but diverges to solve a particular problem. The goal of a software programmer should be to first try understanding existing techniques and then either use or improve them. Solving the same problem over and over without any knowledge of the past is nuts.

Of course, in as much as there are positive patterns that work, there are also negative patterns that have failed over and over down through history. Take personal flight. For many, many years, truly intelligent people tried over and over to strap wings on their arms or backs and fly. They were close in concept, but just doing the same thing over and over was truly folly. Once it was understood how to apply Bernoulli’s principle to building wings and what it would truly take to fly, the Wright Brothers applied this principal, plus principles of lift, to produce the first flying machine. If you ever happen by Kitty Hawk, NC, you can see the plane and location of that flight. Not an amazing amount has changed between that airplane and today’s airplanes in basic principle. Once they got it right, it worked.

In designing and implementing a database, you get the very same sort of things going on. The problem with patterns and anti-patterns is that you don’t want to squash new ideas immediately. The anti-patterns I will present later in this chapter may be very close to something that becomes a great pattern. Each pattern is there to solve a problem, and in some cases, the problem solved isn’t worth the side effects.

Throughout this book so far, we have covered the basic implementation tools that you can use to assemble solutions that meet your real-world needs. In this chapter, I am going to extend this notion and present a few deeper examples where we assemble a part of a database that deals with common problems that show up in almost any database solution. The chapter will be broken up into two major sections. In the first section, we will cover patterns that are common and generally desirable to use. The second half will be anti-patterns, or patterns that you may frequently see that are not desirable to use (along with the preferred method of solution, naturally).
Desirable Patterns

In this section, I am going to cover a good variety of implementation patterns that can be used to solve a number of very common problems that you will frequently encounter. By no means should this be confused with a comprehensive list of the types of problems you may face; think of it instead as a sampling of methods of solving some common problems.

The patterns and solutions that I will present are as follows:

- **Uniqueness**: Moving beyond the simple uniqueness we covered in the first chapters of this book, we’ll look at some very realistic patterns of solutions that cannot be implemented with a simple uniqueness constraint.

- **Data-driven design**: The goal of data driven design is that you never hard-code values that don’t have a fixed meaning. You break down your programming needs into situations that can be based on sets of data values that can be modified without affecting code.

- **Hierarchies**: A very common need is to implement hierarchies in your data. The most common example is the manager-employee relationship. In this section, I will demonstrate the two simplest methods of implementation and introduce other methods that you can explore.

- **Images, documents, and other files**: There is, quite often, a need to store documents in the database, like a web users’ avatar picture, or a security photo to identify an employee, or even documents of many types. We will look at some of the methods available to you in SQL Server and discuss the reasons you might choose one method or another.

- **Generalization**: In this section, we will look at some ways that you will need to be careful with how specific you make your tables so that you fit the solution to the needs of the user.

- **Storing user-specified data**: You can’t always design a database to cover every known future need. In this section, I will cover some of the possibilities for letting users extend their database themselves in a manner that can be somewhat controlled by the administrators.

---

**Note**  
I am always looking for other patterns that can solve common issues and enhance your designs (as well as mine). On my web site (drsql.org), I may make additional entries available over time, and please leave me comments if you have ideas for more.

---

Uniqueness

If you have been reading this book straight through, you’re probably getting a bit sick of hearing about uniqueness. The fact is, uniqueness is one of the largest problems you will tackle when designing a database, because telling two rows apart from one another can be a very difficult task. Most of our efforts so far have been in trying to tell two rows apart, and that is still a very important task that you always need to do.

But, in this section, we will explore a few more types of uniqueness that hit at the heart of the problems you will come across:

- **Selective**: Sometimes, we won’t have all of the information for all rows, but the rows where we do have data need to be unique. As an example, consider the driver’s license numbers of employees. No two people can have the same information, but not everyone will necessarily have one.