CHAPTER 2

Getting It Right: Designing the Database for Performance

By Louis Davidson

Most of the topics in the book are related to admin tasks: indexing, performance tuning, hardware, backup, compression, and so forth. Many times, database administrators (DBAs) don't have input into the design of the database they are responsible for. But even if you don't have control over the design, it's nice to know what a good design ought to look like—if for no other reason than you can be specific in your complaints when you are struggling with a piece of software.

In this chapter, I'll present an overview of the entire process of designing a database system, discussing the factors that make a database perform well. Good performance starts early in the process, well before code is written during the definition of the project. (Unfortunately, it really starts well above of the pay grade of anyone who is likely to read a book like this.) When projects are hatched in the board room, there's little understanding of how to create software and even less understanding of the unforgiving laws of time. Often the plan boils down to “We need software X, and we need it by a date that is completely pulled out of…well, somewhere unsavory.” So the planning process is cut short of what it needs to be, and you get stuck with the seed of a mess. No matter what part you play in this process, there are steps required to end up with an acceptable design that works, and that is what you as a database professional need to do to make sure this occurs.

In this chapter, I’ll give you a look at the process of database design and highlight many of the factors that make the goals of many of the other chapters of this book a whole lot easier to attain. Here are the main topics I’ll talk about:

- **Requirements** Before your new database comes to life, there are preparations to be made. Many database systems perform horribly because the system that was initially designed bore no resemblance to the system that was needed.

- **Table Structure** The structure of the database is very important. Microsoft SQL Server works a certain way, and it is important that you build your tables in a way that maximizes the SQL Server engine.

- **Design Testing** The process of testing should go beyond code and extend into the design process, with the primary goal of the process being to make sure the software that is created matches to needs of the user.
The process of database design is not an overly difficult one, yet it is so often done poorly. Throughout my years of writing, working, speaking, and answering questions in forums, easily 90 percent of the problems came from databases that were poorly planned, poorly built, and (perhaps most importantly) unchangeable because of the mounds of code accessing those structures. With just a little bit of planning and some knowledge of the SQL Server engine’s base patterns, you’ll be amazed at what you can achieve and that you can do it in a lot less time than initially expected.

Requirements

The foundation of any implementation is an understanding of what the heck is supposed to be created. Your goal in writing software is to solve some problem. Often, this is a simple business problem like creating an accounting system, but sometimes it’s a fun problem like shuffling riders on and off of a theme-park ride, creating a television schedule, creating a music library, or solving some other problem. As software creators, our goal ought to be to automate the brainless work that people have to do and let them use their brains to make decisions.

Requirements take a big slap in the face because they are the first step in the classic “waterfall” software-creation methodology. The biggest lie that is common in the programming community is that the waterfall method is completely wrong. The waterfall method states that a project should be run in the following steps:

- **Requirements Gathering** Document what a system is to be, and identify the criteria that will make the project a success.
- **Design** Translate the requirements into a plan for implementation.
- **Implementation** Code the software.
- **Testing** Verify that the software does what it is supposed to do.
- **Maintenance** Make changes to address problems not caught in testing.
- **Repeat the process.**

The problem with the typical implementation of the waterfall method isn’t the steps, nor is it the order of the steps, but rather it’s the magnitude of the steps. Projects can spend months or even years gathering requirements, followed by still more months or years doing design. After this long span of time, the programmers finally receive the design to start coding from. (Generally, it is slid under their door so that the people who devised it can avoid going into the dungeons where the programmers are located, shackled to their desks.) The problem with this approach is that, the needs of the users changed frequently in the years that passed before software was completed. Or (even worse) as programming begins, it is realized that the requirements are wrong, and the process has to start again.

As an example, on one of my first projects as a consultant, we were designing a system for a chemical company. A key requirement we were given stated something along the lines of: “Product is only sold when the quality rating is not below 100.” So, being the hotshot consultant programmer who wanted to please his bosses and customers, I implemented the database to prevent shipping the product when the rating was 99.9999 or less, as did the UI programmer. About a week after the system is shipped, the true requirement was learned. “Product is only sold when the quality rating is not below 100…or the customer overrides the rating because they want to.” D’oh! So after a crazy few days where sleep was something we only dreamt about, we corrected the issues. It was an excellent life lesson, however. Make sure requirements make sense before programming them (or at least get it down in writing that you made sure!)

As the years have passed and many projects have failed, the pendulum has swung away from the pure waterfall method of spending years planning to build software, but too often the opposite now occurs. As a reaction to the waterfall method, a movement known as Agile has arisen. The goal of Agile is to attempt to