The Database is King October 25, 2017

Over the last twenty plus years, I've been involved in numerous software development projects. All of these projects have been different in many ways, but they all had one common element – the extensive use of a database. If I look back at all of the projects I've been involved with, I'm able to draw some conclusions with respect to why certain projects succeeded, while others failed.

Without a doubt, the single most important factor that determines the success or failure of a project is the database. More specifically, there are two aspects of the database that are important. The first is the overall quality of the database design (proper table structures, good use of indexes, etc.) Obviously, this is crucial. A poor database design can create massive problems, both in the short term and long term. In addition, even if a database is mostly solid (but suboptimal), it almost always creates headaches caused by the necessity of workarounds, hacks, etc.

In addition to the quality of the database design, the second, and more important aspect of the database, is the priority given to it. Let's repeat that for emphasis: The most important thing successful projects have in common is the priority given to the database.

Back in the "old" days (before the rise of ORM), making the database a priority was a given, and most project leaders would try to find a good DBA (or a lead developer/architect with high-level database skills) before hiring anyone else on a team. This has changed somewhat over the years due to a couple of different phenomena. One is the rise in popularity of ORM solutions. I've commented previously on ORM (See "ORM is an Anti-Pattern"). Without going into detail about that subject again, I will make one additional point, mostly because I find it amusing. I had lunch with a colleague recently (a guy who also has more than 20 years of experience) and we agreed that ORM wasn't just an anti-pattern, it was more akin to cancer. Once ORM infiltrates a project, it tends to metastasize and affect all aspects of a project in an extremely deleterious way (performance, maintainability, debuggability, etc.). Not only that, ORM tumors are very hard to eradicate, particularly if an entire application has been architected around it. Indeed, after being tasked to fix a few ORM messes, I've concluded that you're probably better off doing a rewrite, rather than hoping that chemotherapy or brute surgical force will alleviate your problems. ORM creates a convoluted foundation of sand; a foundation that simply won't provide strong long-term support for an application.

The second thing that has changed in the software development landscape is the related fad of code-first development. The idea is that it makes more sense to design the business objects around the requirements of your domain and then worry about the database implementation of those objects later. Makes sense, right? Shouldn't business objects be designed tightly around the business requirements of the domain? While this idea has some superficial appeal, I reject it for several reasons.

First, as I've stated previously, I spurn the idea that OO objects are more important than the database. In other words, the design of the database usually shouldn't bend in order to achieve "good" OO design, particularly since these two things can easily co-exist. The overarching point is that nobody cares about the purity of your business objects, or how well they conform to the OO textbook. People care about data, and the data is where all of the value lies.

Second, databases often serve multiple masters. What I mean is: you may have designed (or failed to design) a database for a single application, but down the road, the data needs to be used by other applications, or integrate with other systems, or support new front ends, etc. If you've taken database design "shortcuts" in order to serve the OO domain design master, you may end in a very painful place.

Finally, let's address the debuggability of applications. Needless to say, being able to easily debug any application is extremely important. Otherwise, you can spend hours and hours and hours trying to diagnose issues. In my career, I've been hired many times to fix/debug large, complex applications. Often, it is necessary to find/isolate very specific pieces of data in order to solve a difficult bug. Sometimes, you have to write complex SQL, or create test data or temporary database objects to get to the root of problems. This often requires high-level SQL skills. Personally, I've found that developers who lack solid database/SQL skills are severely handicapped by this lack of expertise. So much so, in fact, that I only hire developers who have outstanding database skills and a keen understanding of why the database is king.