ORM is an Anti-Pattern

April 2, 2017

A large number of articles have been written that explain why Object Relational Mapping (ORM) is an anti-pattern. I generally agree with all of those articles, which cover a wide array of reasons that ORM has no place in software applications, large or small.

Since so many articles have been written on this subject, why write another? I don't really have anything new to add, other than my own experiences with it, as well as a strong philosophical hatred of it. That being said, I want to be on the record publicly against ORM, since I think it is an important issue that can literally be application destroying. It also makes my life miserable every time I have to fix it and educate other developers about why it is such a bad idea.

Over the course of twenty years in the software industry, I've had the distinct displeasure of dealing with ORM three times. The first time I was given the task of fixing some horrendous NHibernate performance problems. I spent several months stepping through bloated NHibernate code, while paying close attention to SQL Profiler so that I could observe first-hand, the bloated and inefficient SQL generated by the NHibernate engine. In the end, my manager told me to rip out the NHibernate code and replace it with highly optimized stored procedures. Needless to say, the new code was more readable, maintainable, and performed orders of magnitude faster.

Usually, when people choose ORM, it is for one of two reasons: 1) Developers think they can save development time by letting ORM handle database operations, particularly CRUD operations. While there might be a tiny kernel of truth to this in very small applications (prototypes maybe), 99% of applications outgrow ORM very quickly, and any time savings are blown out of the water by having to fix performance and other issues related to software bloat. 2) Developers don't know SQL and don't want to learn it. This was the reason for the use of NHibernate in a project that I was recently hired to fix. To make a long story short, due to egregious performance problems and bloated unreadable code, we had little choice but to replace ORM with optimized SQL.

In addition to the "practical" reasons developers choose NHibernate, there also appears to be a rather hideous undercurrent of faddish hipsterism, most pronounced when ORM first burst on to the scene. This toxin still exists to some extent, although it doesn't appear to be as unhealthy in the .NET community (Java, by contrast, has been infected by ORM for much longer).

Speaking of ORM advocates, the dogma they usually spout is that ORM solves the "impedance mismatch" problem, which, in short, is the "problem" that a database is (usually) relational while OO objects are not. The truth is that this problem doesn't really exist, and even if it did, ORM doesn't "solve" it. The only thing ORM does is make dealing with complex applications less flexible and more complex than they need to be. The other thing to note is the implication that databases need to be "fixed" in order to conform to the dictates of object-oriented programming. While OO is certainly valuable and has an important place in software development, the database doesn't need to be shoe-horned into artificial and contrived objects, just for the sake of some architect's theoretical hangups. At the end of the day, databases answer questions, and software design needs to revolve around answering those questions correctly and efficiently, even if that means deviating from the college textbook on programming.

Finally, there's been a disturbing trend in software development over the last several years - the idea of "code first" development. The basics of this idea is that you design and develop the business objects first, and then you create the database, often using an automated "database creator." Without going into detail about why this doesn't work well, my primary objections to it are more philosophical. In short, nothing really matters but the data. The data are the (valuable) jewels, and people need software developers to safeguard data and make it accessible and usable in an efficient way. Indeed, a developer's first priority must be on protecting the integrity of data and making it as efficient as possible to access. ORM and other "code first" approaches fail. As such, they should be categorically rejected in favor of more thoughtful design patterns.