The Database is Queen February 8, 2018

A few months ago, I wrote a blog about the importance of making the database the top priority in any web application (See "The Database is King"). I have a few more things to say about that, and in the interest of gender fairness, I decided to write a little more on this topic.

Over the past twenty plus years, I've seen many databases with issues. I've also seen many mistakes made. Issues and mistakes don't bother me at all because mistakes are made every day in software development. It's unavoidable, so you usually just fix them and move on. What I've always found surprising is that people will tolerate a crappy, sub-optimal, difficult-to-understand database because it is too much "trouble" to fix.

People's objections to fixing the database are many. I often hear things like:

"If we change the database, we'd have to make too many changes across the application."

"There are too many dependencies that would break if we changed the database."

"Does it really matter if we allow nulls in this column? It's never going to be null, so don't worry about it."

"The field names don't make sense, but everyone knows what they are."

"A database change would require updating too many existing records."

"A few orphaned records aren't going to hurt anything."

"Normalizing/optimizing the database wouldn't match our object model."

"We don't have time to fix the database. We're too busy trying to get ORM to work."

"The database implementation doesn't really matter. We have business objects on the top of the database which fix (mask) the underlying problems."

When there's a problem with the database (even a relatively small one), it should be fixed. "But…" No! Fixed! Why do I care so much about this? I've spent a lot of time dealing with data issues in complex applications. I usually don't mind doing this, and, most of the time, I enjoy doing it. What isn't enjoyable, however, is when I have to navigate a database where people refuse to fix issues – issues that make the database more difficult to query, more difficult to understand, and, in the worst cases, just simply unreliable.

I was on a project recently where there was a serious data issue; so serious that it wasn't clear if some of the data was valid. As I investigated the issue, it became immediately obvious that the issue would not have existed if the tables had been properly structured and the relationships had been properly defined. In other words, the problem would have revealed itself immediately because the database would've thrown an error and not allowed the "bad data” condition. Because the tables weren’t structured properly, it was a very difficult and time-consuming problem to debug.

Speaking of debugging, I think it’s fair to say that code problems are much, much easier to debug than data problems. Usually, code problems reveal themselves through an application crash, or can be identified through rudimentary error handling. Often, it’s just a matter of stepping through the code and isolating the problem. Data issues are a different animal, especially if they occur in a database that isn’t properly structured. These can be your worst nightmare.

Did I mention that database problems should be fixed immediately? “But…” No! Fixed!

As I’ve said before, the database is king. It is also the queen, and, for that matter, the entire royal court (except, of course, the jester, who is that foolish ORM hipster that doesn’t want to hear any of this). Treat the database with the proper respect.