**[PolyglotPersistence](https://martinfowler.com/bliki/PolyglotPersistence.html)**

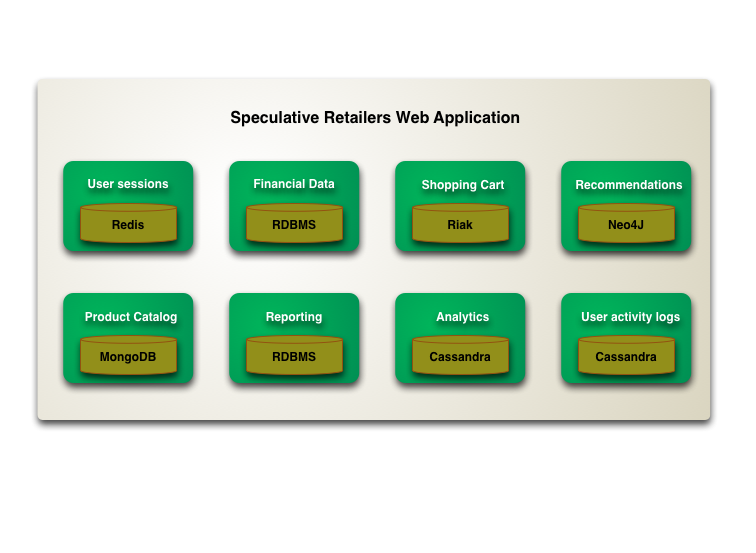
*[](https://martinfowler.com/)*

[*Martin Fowler*](https://martinfowler.com/)

*16 November 2011*

In 2006, my colleague Neal Ford coined the term [Polyglot Programming](http://memeagora.blogspot.com/2006/12/polyglot-programming.html), to express the idea that applications should be written in a mix of languages to take advantage of the fact that different languages are suitable for tackling different problems. Complex applications combine different types of problems, so picking the right language for the job may be more productive than trying to fit all aspects into a single language.

Over the last few years there's been an explosion of interest in new languages, particularly functional languages, and I'm often tempted to spend some time delving into Clojure, Scala, Erlang, or the like. But my time is limited and I'm giving a higher priority to another, more significant shift, that of the [DatabaseThaw](https://martinfowler.com/bliki/DatabaseThaw.html). The first drips have been coming through from clients and other contacts and the prospects are enticing. I'm confident to say that if you starting a new strategic enterprise application you should no longer be assuming that your persistence should be relational. The relational option might be the right one - but you should seriously look at other alternatives.

**

One of the interesting consequences of this is that we are gearing up for a shift to polyglot persistence [[1]](https://martinfowler.com/bliki/PolyglotPersistence.html#footnote-first-use) - where any decent sized enterprise will have a variety of different data storage technologies for different kinds of data. There will still be large amounts of it managed in relational stores, but increasingly we'll be first asking how we want to manipulate the data and only then figuring out what technology is the best bet for it.

This polyglot affect will be apparent even within a single application[[2]](https://martinfowler.com/bliki/PolyglotPersistence.html#footnote-example). A complex enterprise application uses different kinds of data, and already usually integrates information from different sources. Increasingly we'll see such applications manage their own data using different technologies depending on how the data is used. This trend will be complementary to the trend of breaking up application code into separate components integrating through web services. A component boundary is a good way to wrap a particular storage technology chosen for the way its data in manipulated.

This will come at a cost in complexity. Each data storage mechanism introduces a new interface to be learned. Furthermore data storage is usually a performance bottleneck, so you have to understand a lot about how the technology works to get decent speed. Using the right persistence technology will make this easier, but the challenge won't go away.

Many of these NoSQL option involve running on large clusters. This introduces not just a different data model, but a whole range of new questions about consistency and availability. The transactional single point of truth will no longer hold sway (although its role as such has often been illusory).

So polyglot persistence will come at a cost - but it will come because the benefits are worth it. When relational databases are used inappropriately, they exert a significant drag on application development. I was recently talking to a team whose application was essentially composing and serving web pages. They only looked up page elements by ID, they had no need for transactions, and no need to share their database. A problem like this is much better suited to a key-value store than the corporate relational hammer they had to use. A good public example of using the right NoSQL choice for the job is The Guardian - who have felt a definite productivity gain from using MongoDB over their previous relational option.

Another benefit comes in running over a cluster. Scaling to lots of traffic gets harder and harder to do with vertical scaling - a fact we've known for a long time. Many NoSQL databases are designed to operate over clusters and can tackle larger volumes of traffic and data than is realistic with single server. As enterprises look to use data more, this kind of scaling will become increasingly important. The Danish medication system described at [gotoAarhus2011](https://martinfowler.com/bliki/gotoAarhus2011.html) was a good example of this.

All of this leads to a big change, but it won't be rapid one - companies are naturally conservative when it comes to their data storage.

The more immediate question is which types of projects should consider an alternative persistence model? My thinking is that firstly you should only consider projects that are at the strategic end of the [UtilityVsStrategicDichotomy](https://martinfowler.com/bliki/UtilityVsStrategicDichotomy.html). That's because utility projects don't have enough benefit to be worth a new technology.

Given a strategic project, you then have two drivers that raise alternatives: either reducing development drag or dealing with intensive data needs. Even here I suspect many projects, probably a majority, are better off sticking with the relational orthodoxy. But the minority that shouldn't is a significant one.

One factor that is perhaps less important is whether the project is new, or already established. The Guardian's shift to MongoDB has been happening over the last year or so on a code base developed several years ago. Polyglot persistence is something you can introduce on an existing code base.

What all of this means is that if you're working in the enterprise application world, now is the time to start familiarizing yourself with alternative data storage options. This won't be a fast revolution, but I do believe the next decade will see the database thaw progress rapidly.

**Notes**

**1:**As far as I can tell, Scott Leberknight was the first person to [start using the term](http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence) "polyglot persistence".

**2:**Don't take the example in the diagram too seriously. I'm not making any recommendations about which database technology to use for what kind of service. But I do think that people should consider these kinds of technologies as part of application architecture.