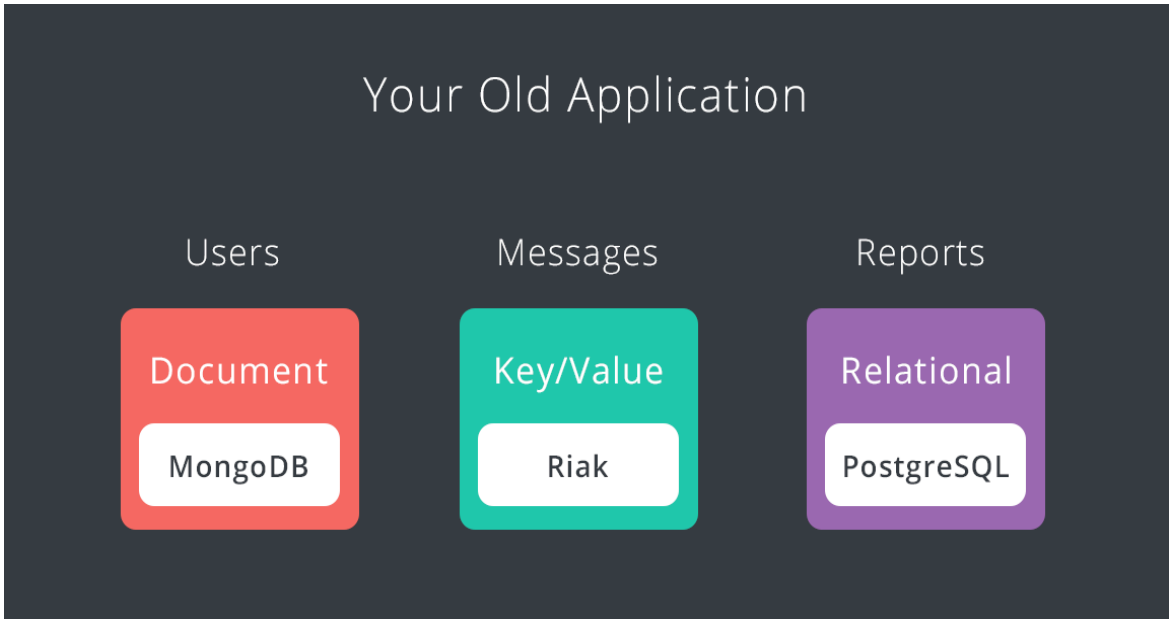
Polyglot Persistence or Multiple Data Models?

OCT 28, 2013 – BY STEPHEN PIMENTEL

The rise of distributed databases, spanning both NoSQL and older relational technologies, has produced all the confusion typical of fast-moving fields whose engineering designs have not yet matured. Developers now face many choices when deciding how to store and manage their data, although the available solutions differ greatly in their performance, scalability, and flexibility.

One aspect of database architecture that has not received enough attention is the tight coupling of three different levels of the technology that makes up a database. When you choose most databases, you're not choosing just one technology, you're choosing three: a storage engine, data model, and query language. For example, if you choose Postgres, you're choosing the Postgres storage engine, a relational data model, and SQL. If you choose MongoDB, you're choosing MongoDB's BSON storage, a document data model, and the MongoDB API.



Relational databases, of course, support the relational data model, generally with SQL as query language. In a similar way, the first-generation NoSQL

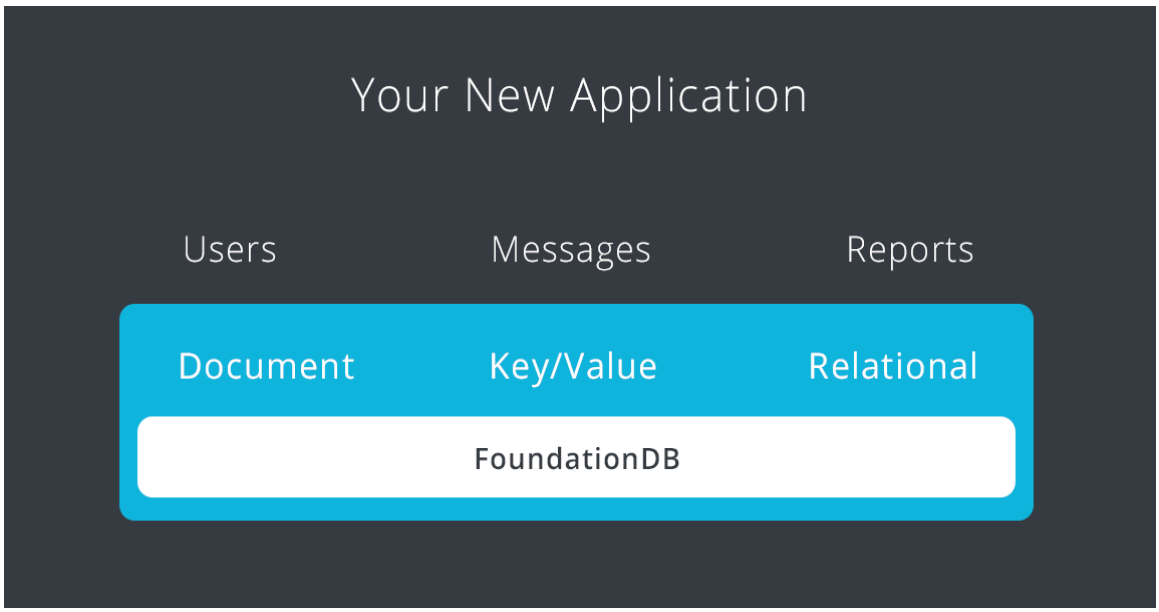
databases each support a single data model, such as a document, graph, or column-oriented model, along with a specialized query language.

In these systems, features tend to be interwoven among all the levels, with the data model hard-wired in the middle. The restriction to a single data model limits the range of use cases the database can handle well: different data models are better suited to different use cases, and applications often require more than one data model for different data types.

As a result, developers face a choice between shoehorning all of their application's data into an inappropriate model and integrating multiple databases into their application's backend. Calling this option "polyglot persistence" makes it sound rather sophisticated. The reality is otherwise. At best, the need to integrate multiple databases imposes a significant engineering and operational cost - the team needs to have experts in each database technology on hand, and just as importantly, in order for the application to stay online all of the databases need to remain up. This makes the fault tolerance of the application equal to the weakest database in the stack. Too often, this devolves into an operational nightmare.

In an often-cited post on polyglot persistence, Martin Fowler sketches a web application for a hypothetical retailer that uses each of Riak, Neo4j, MongoDB, Cassandra, and an RDBMS for distinct data sets. It's not hard to imagine his retailer's DevOps engineers quitting in droves. Even if most realistic architectures are less extreme, the truth is that polyglot persistence is a workaround for data modeling problems, not a solution to them. Developers need to customize data models for an application and often need more than one, but they shouldn't have to adopt different databases to get them.

Rather than having to integrate multiple databases, it's far better if a developer can build multiple data models easily and efficiently on a single storage substrate. The key feature that makes this approach possible is ACID transactions with high performance and scalability. Building a custom data model that supports concurrent updates usually requires synchronization of disparate data elements. Such synchronization is easy with ACID transactions and very difficult without them.



For example, imagine having to remove a user and their related data in the message and report models. Without ACID, your application might have a lot of code to handle the case where a message references a user that no longer exists (because a read arrived in between the writes to delete the user and then delete their messages). With ACID, you can safely remove all user data in a single transaction and thus removing the need for the aforementioned application code.

High-performance, scalable transactions facilitate an architecture in which a storage substrate is combined with a variety of layers, allowing developers to select the data models best suited to their use cases.

The need for multiple data models in complex, modern applications is a reality. Polyglot persistence attempts to respond to this challenge but imposes further operational complexity and cost. A better solution is for a single database to provide flexible support for multiple data models, while meeting the growing requirements for fault tolerance, scalability, and performance.

— Stephen Pimentel