Multimodel Database or Polyglot Persistence

Abhishek Tiwari

Citation: *A. Tiwari*, "Multimodel Database or Polyglot Persistence", Abhishek Tiwari, 2015. doi:10.59350/6jena-v1040

Published on: March 10, 2015

In a recent InfoWorld article, FoundationDB's Stephen Pimentel explained the rise of the multimodel database. A multimodel database uses a single data store to support multiple data models. For instance, FoundationDB tries to map documents, graphs, and relational tables to a key-value data store. Applications can either send their data directly to the FoundationDB key-value data store or they can speak to co-located layers representing multiple data models. Under the hood a layer is stateless and translates a data model to store them as key-value store.

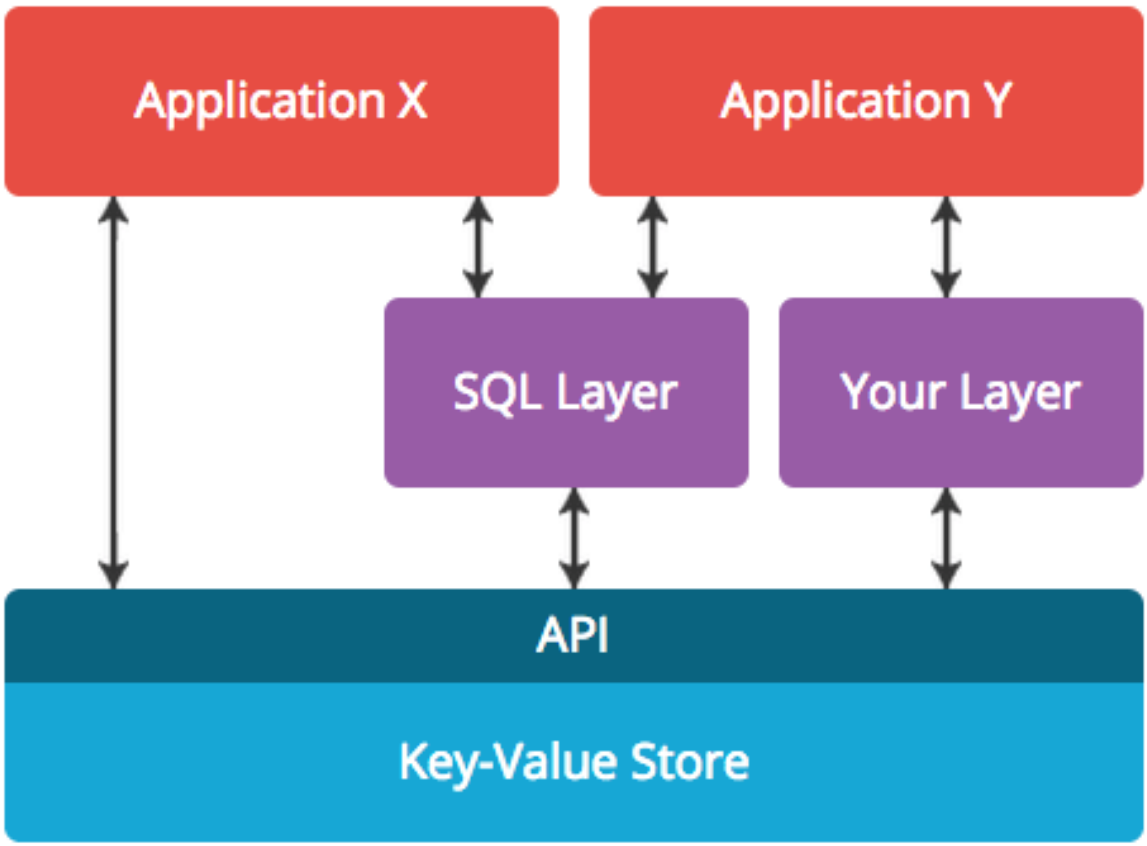

Figure 1: FoundationDB is a multimodel database

A multimodel database is quite different from polyglot persistence which support multiple data models using multiple data stores. Although polyglot persistence is a highly effective architecture but not without limitations. With polyglot persistence one may end up with multiple databases (both SQL and NoSQL), each with its own storage and operational requirements. Unlike polyglot persistence where you have to manage fault tolerance, scalability, and performance requirements for multiple data stores individually, with multimodel database these requirements are simplified due to single data store. That said, this may not be an issue if your requirements can be met using Database-as-a-Service (DaaS) offerings. For majority of use cases DaaS offerings such AWS RDS, AWS DynamoDB,

AWS Elasticache Redis, etc will be more than sufficient. In addition, with multimodel database data integration can be easier when compared to polyglot persistence.

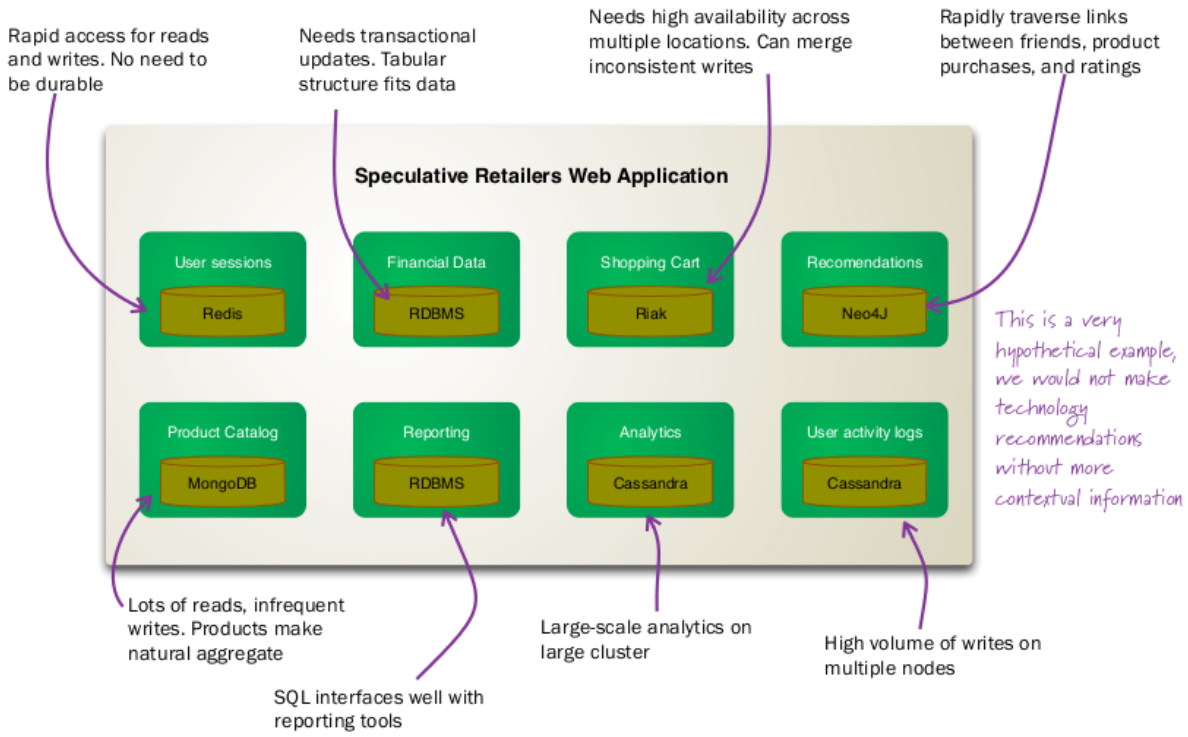

Figure 2: Polyglot persistence - choice to select best data store for a given data model

In a nutshell, a multimodel database allows to keep good parts of polyglot persistence (i.e. support for multiple data models) but helps to lose the bad parts (i.e. reduced requirements for fault tolerance, scalability, and performance).

Update-1: Week after I published this article coincidently Apple acquired the FoundationDB and basically killed the the whole FoundationDB platform. FoundationDB is not available for download anymore, all their public Github repos are unaccessible. Future of existing FoundationDB customers and users is uncertain at this stage.

Update-2: Commenting on FoundationDB acquisition, John Hugg of VoltDB wrote why a fast key-value store is not enough

> fast data mutation and lookup-by-key are important, but they're not enough. At high mutation rates, users want visibility into how their data is changing and how their business is affected, usually in real-time. A fast key-value store - even one with strong ACID properties - is all well and good, but without a powerful query language and real index support, it's much less interesting.