[redacted]

NoSQL Distilled

This page intentionally left blank

NoSQL Distilled

A Brief Guide to the Emerging World of Polyglot Persistence

Pramod J. Sadalage
Martin Fowler

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382–3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Sadalage, Pramod J.
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-82662-6 (pbk. : alk. paper) -- ISBN 0-321-82662-0 (pbk. : alk. paper) 1. Databases--Technological innovations. 2. Information storage and retrieval systems. I. Fowler, Martin, 1963- II. Title.
QA76.9.D32S228 2013
005.74--dc23

ISBN-13: 978-0-321-82662-6
ISBN-10: 0-321-82662-0
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
Fourth printing, November 2014

For my teachers Gajanan Chinchwadkar, Dattatraya Mhaskar, and Arvind Parchure. You inspired me the most, thank you.

—Pramod

For Cindy

—Martin

This page intentionally left blank

This page intentionally left blank

Had we not thought that, we wouldn't have spent the time and effort writing this book.

This book seeks to give you enough information to answer the question of whether NoSQL databases are worth serious consideration for your future projects. Every project is different, and there's no way we can write a simple decision tree to choose the right data store. Instead, what we are attempting here is to provide you with enough background on how NoSQL databases work, so that you can make those judgments yourself without having to trawl the whole web. We've deliberately made this a small book, so you can get this overview pretty quickly. It won't answer your questions definitively, but it should narrow down the range of options you have to consider and help you understand what questions you need to ask.

Why Are NoSQL Databases Interesting?

We see two primary reasons why people consider using a NoSQL database.

- Application development productivity. A lot of application development effort is spent on mapping data between in-memory data structures and a relational database. A NoSQL database may provide a data model that better fits the application's needs, thus simplifying that interaction and resulting in less code to write, debug, and evolve.
- Large-scale data. Organizations are finding it valuable to capture more data and process it more quickly. They are finding it expensive, if even possible, to do so with relational databases. The primary reason is that a relational database is designed to run on a single machine, but it is usually more economic to run large data and computing loads on clusters of many smaller and cheaper machines. Many NoSQL databases are designed explicitly to run on clusters, so they make a better fit for big data scenarios.

What's in the Book

We've broken this book up into two parts. The first part concentrates on core concepts that we think you need to know in order to judge whether NoSQL databases are relevant for you and how they differ. In the second part we concentrate more on implementing systems with NoSQL databases.

Chapter 1 begins by explaining why NoSQL has had such a rapid rise—the need to process larger data volumes led to a shift, in large systems, from scaling vertically to scaling horizontally on clusters. This explains an important feature of the data model of many NoSQL databases—the explicit storage of a rich structure of closely related data that is accessed as a unit. In this book we call this kind of structure an *aggregate*.

Chapter 2 describes how aggregates manifest themselves in three of the main data models in NoSQL land: key-value (“Key-Value and Document Data Models,” p. 20), document (“Key-Value and Document Data Models,” p. 20), and column family (“Column-Family Stores,” p. 21) databases. Aggregates provide a natural unit of interaction for many kinds of applications, which both improves running on a cluster and makes it easier to program the data access. Chapter 3 shifts to the downside of aggregates—the difficulty of handling relationships (“Relationships,” p. 25) between entities in different aggregates. This leads us naturally to graph databases (“Graph Databases,” p. 26), a NoSQL data model that doesn’t fit into the aggregate-oriented camp. We also look at the common characteristic of NoSQL databases that operate without a schema (“Schemaless Databases,” p. 28)—a feature that provides some greater flexibility, but not as much as you might first think.

Having covered the data-modeling aspect of NoSQL, we move on to distribution: Chapter 4 describes how databases distribute data to run on clusters. This breaks down into sharding (“Sharding,” p. 38) and replication, the latter being either master-slave (“Master-Slave Replication,” p. 40) or peer-to-peer (“Peer-to-Peer Replication,” p. 42) replication. With the distribution models

Who Should Read This Book

Our target audience for this book is people who are considering using some form of a NoSQL database. This may be for a new project, or because they are hitting barriers that are suggesting a shift on an existing project.

Our aim is to give you enough information to know whether NoSQL technology makes sense for your needs, and if so which tool to explore in more depth. Our primary imagined audience is an architect or technical lead, but we think this book is also valuable for people involved in software management who want to get an overview of this new technology. We also think that if you're a developer who wants an overview of this technology, this book will be a good starting point.

We don't go into the details of programming and deploying specific databases here—we leave that for specialist books. We've also been very firm on a page limit, to keep this book a brief introduction. This is the kind of book we think you should be able to read on a plane flight: It won't answer all your questions but should give you a good set of questions to ask.

If you've already delved into the world of NoSQL, this book probably won't commit any new items to your store of knowledge. However, it may still be useful by helping you explain what you've learned to others. Making sense of the issues around NoSQL is important—particularly if you're trying to persuade someone to consider using NoSQL in a project.

What Are the Databases

In this book, we've followed a common approach of categorizing NoSQL databases according to their data model. Here is a table of the four data models and some of the databases that fit each model. This is not a comprehensive list—it only mentions the more common databases we've come across. At the time of writing, you can find more comprehensive lists at http://nosql-database.org and http://nosql.mypopescu.com/kb/nosql. For each category, we mark with italics the database we use as an example in the relevant chapter.

Our goal is to pick a representative tool from each of the categories of the databases. While we talk about specific examples, most of the discussion should apply to the entire category, even though these products are unique and cannot be generalized as such. We will pick one database for each of the key-value, document, column family, and graph databases; where appropriate, we will mention other products that may fulfill a specific feature need.

Data Model	Example Databases
Key-Value ("Key-Value Databases," p. 81)	BerkeleyDB LevelDB Memcached Project Voldemort Redis *Riak*
Document ("Document Databases," p. 89)	CouchDB *MongoDB* OrientDB RavenDB Terrastore
Column-Family ("Column-Family Stores," p. 99)	Amazon SimpleDB *Cassandra* HBase Hypertable
Graph ("Graph Databases," p. 111)	FlockDB HyperGraphDB Infinite Graph *Neo4J* OrientDB

This classification by data model is useful, but crude. The lines between the different data models, such as the distinction between key-value and document databases ("Key-Value and Document Data Models," p. 20), are often blurry. Many databases don't fit cleanly into categories; for example, OrientDB calls itself both a document database and a graph database.

Acknowledgments

Our first thanks go to our colleagues at ThoughtWorks, many of whom have been applying NoSQL to our delivery projects over the last couple of years. Their experiences have been a primary source both of our motivation in writing this book and of practical information on the value of this technology. The positive

This page intentionally left blank

Chapter 13

Polyglot Persistence

Different databases are designed to solve different problems. Using a single database engine for all of the requirements usually leads to non- performant solutions; storing transactional data, caching session information, traversing graph of customers and the products their friends bought are essentially different problems. Even in the RDBMS space, the requirements of an OLAP and OLTP system are very different—nonetheless, they are often forced into the same schema.

Let's think of data relationships. RDBMS solutions are good at enforcing that relationships exist. If we want to discover relationships, or have to find data from different tables that belong to the same object, then the use of RDBMS starts being difficult.

Database engines are designed to perform certain operations on certain data structures and data amounts very well—such as operating on sets of data or a store and retrieving keys and their values really fast, or storing rich documents or complex graphs of information.

13.1 Disparate Data Storage Needs

Many enterprises tend to use the same database engine to store business transactions, session management data, and for other storage needs such as reporting, BI, data warehousing, or logging information (Figure 13.1).

The session, shopping cart, or order data do not need the same properties of availability, consistency, or backup requirements. Does session management storage need the same rigorous backup/recovery strategy as the e-commerce orders data? Does the session management storage need more availability of an instance of database engine to write/read session data?

In 2006, Neal Ford coined the term polyglot programming, to express the idea that applications should be written in a mix of languages to take advantage

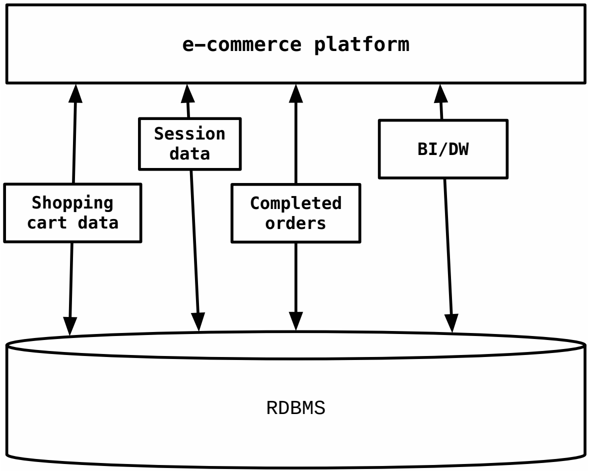


Figure 13.1 *Use of RDBMS for every aspect of storage for the application*

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 each job may be more productive than trying to fit all aspects into a single language.

Similarly, when working on an e-commerce business problem, using a data store for the shopping cart which is highly available and can scale is important, but the same data store cannot help you find products bought by the customers' friends—which is a totally different question. We use the term polyglot persistence to define this hybrid approach to persistence.

13.2 Polyglot Data Store Usage

Let's take our e-commerce example and use the polyglot persistence approach to see how some of these data stores can be applied (Figure 13.2). A key-value data store could be used to store the shopping cart data before the order is confirmed by the customer and also store the session data so that the RDBMS is not used for this transient data. Key-value stores make sense here since the shopping cart is usually accessed by user ID and, once confirmed and paid by the customer, can be saved in the RDBMS. Similarly, session data is keyed by the session ID.

If we need to recommend products to customers when they place products into

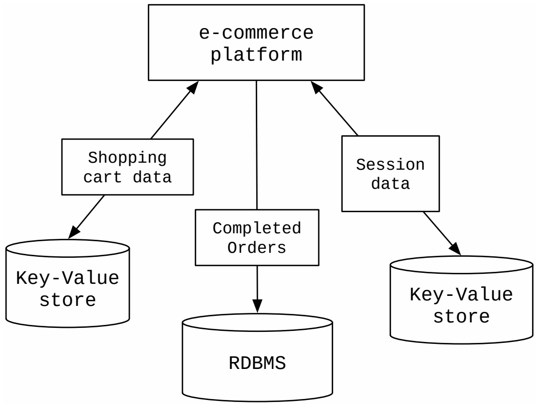


Figure 13.2 *Use of key-value stores to offload session and shopping cart data storage*

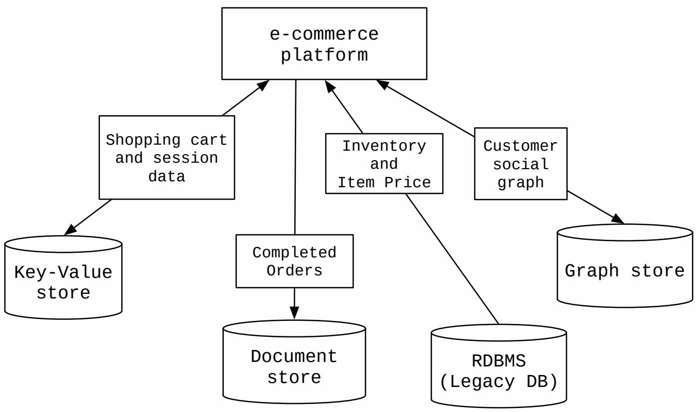


Figure 13.3 *Example implementation of polyglot persistence*

or "your friends bought these accessories for this product"—then introducing a graph data store in the mix becomes relevant (Figure 13.3).

It is not necessary for the application to use a single data store for all of its needs, since different databases are built for different purposes and not all problems can be elegantly solved by a singe database.

Even using specialized relational databases for different purposes, such as data warehousing appliances or analytics appliances within the same application, can be viewed as polyglot persistence.

13.3 Service Usage over Direct Data Store Usage

As we move towards multiple data stores in the application, there may be other applications in the enterprise that could benefit from the use of our data stores or the data stored in them. Using our example, the graph data store can serve data to other applications that need to understand, for example, which products are being bought by a certain segment of the customer base.

Instead of each application talking independently to the graph database, we can wrap the graph database into a service so that all relationships between the nodes can be saved in one place and queried by all the applications (Figure 13.4). The data ownership and the APIs provided by the service are more useful than a single application talking to multiple databases.

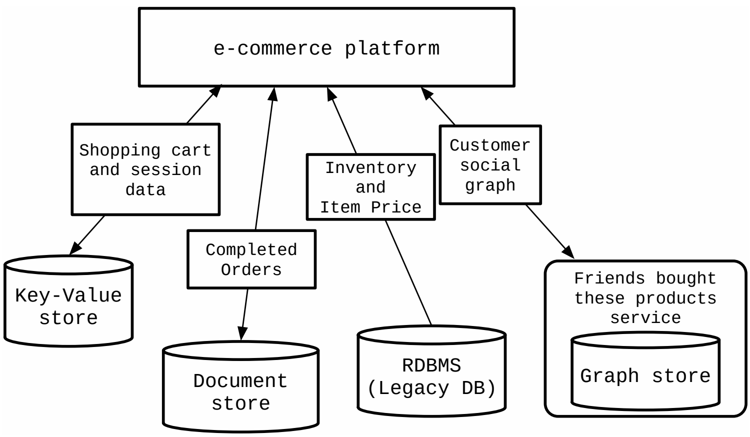


Figure 13.4 *Example implementation of wrapping data stores into services*

The philosophy of service wrapping can be taken further: You could wrap all databases into services, letting the application to only talk to a bunch of services (Figure 13.5). This allows for the databases inside the services to evolve without you having to change the dependent applications.

Many NoSQL data store products, such as Riak [Riak] and Neo4J [Neo4J], actually provide out-of-the-box REST API's.

13.4 Expanding for Better Functionality

Often, we cannot really change the data storage for a specific usage to something different, because of the existing legacy applications and their dependency on

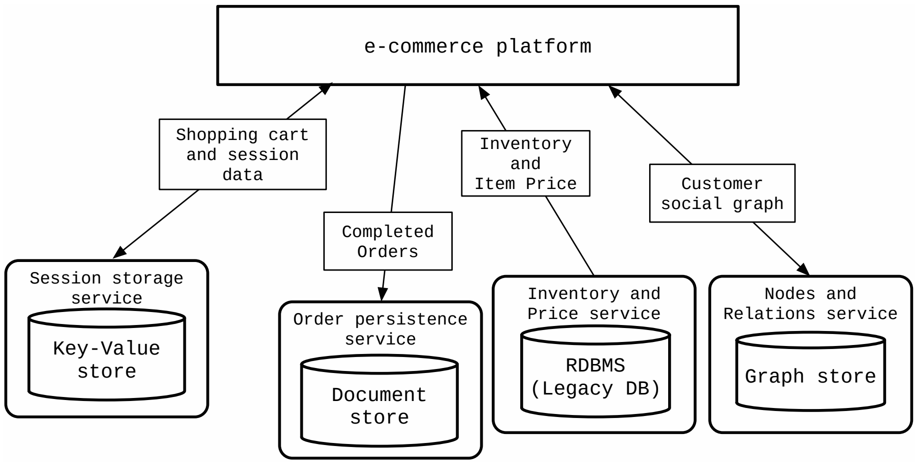
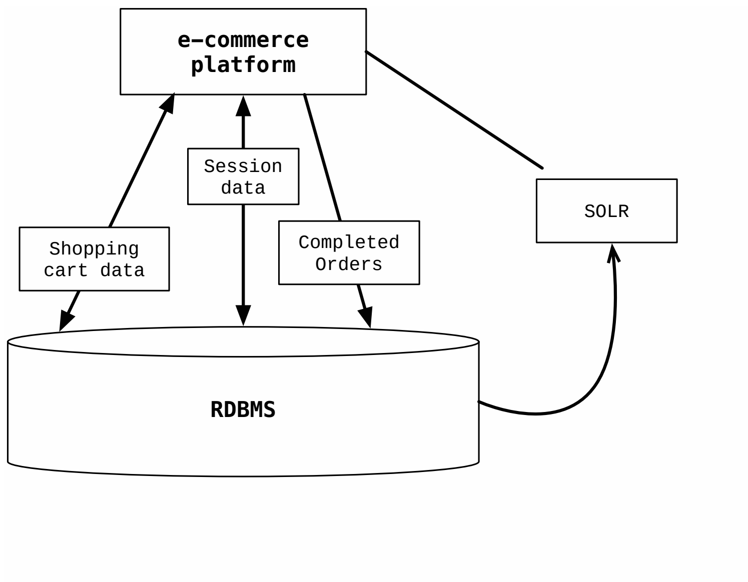


Figure 13.5 *Using services instead of talking to databases*

existing data storage. We can, however, add functionality such as caching for better performance, or use indexing engines such as Solr [Solr] so that search can be more efficient (Figure 13.6). When technologies like this are introduced, we have to make sure data is synchronized between the data storage for the application and the cache or indexing engine.



While doing this, we need to update the indexed data as the data in the application database changes. The process of updating the data can be real-time or batch, as long as we ensure that the application can deal with stale data in the

13.8 Key Points

- Polyglot persistence is about using different data storage technologies to handle varying data storage needs.
- Polyglot persistence can apply across an enterprise or within a single application.
- Encapsulating data access into services reduces the impact of data storage choices on other parts of a system.
- Adding more data storage technologies increases complexity in programming and operations, so the advantages of a good data storage fit need to be weighed against this complexity.

This page intentionally left blank

A

E

F

G

S

T