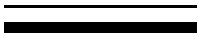


# NoSQL Distilled

*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

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

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](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/aw](http://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

Copyright © 2013 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

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*

Preface .....	xiii
Part I: Understand .....	1
Chapter 1: Why NoSQL .....	3
1.1 The Value of NoSQL Databases .....	2
1.2 The Evolution of NoSQL .....	3
1.3 The Evolution of NoSQL .....	4
1.4 Attack of the Polyglot Persistence .....	8
1.5 The Emergence of New NoSQL Databases .....	9
1.6 Key Points .....	12
Chapter 2: Aggregate Data .....	14



3.3	Schemaless Databases .....	28
3.4	Materialized Views .....	30
3.5	Modeling for Data Access .....	31
3.6	Key Points .....	36
Chapter 4:	Distribution Models .....	37
4.1	Single Server .....	37
4.2	Sharding .....	38
4.3	Master-Slave Replication .....	40
4.4	Peer-to-Peer Replication .....	42
4.5	Combining Sharding and Replication .....	43
4.6	Key Points .....	44
Chapter 5:	Consistency .....	47
5.1	Update Consistency .....	47
5.2	Read Consistency .....	49
5.3	Relaxing Consistency .....	52
5.3.1	<i>The CAP Theorem</i> .....	53
5.4	Relaxing Durability .....	56
5.5	Quorums .....	57
5.6	Further Reading .....	59
5.7	Key Points .....	59
Chapter 6:	Version Stamps .....	61
6.1	Business and System Transactions .....	61
6.2	Version Stamps on Multiple Nodes .....	63
6.3	Key Points .....	65
Chapter 7:	Map-Reduce .....	67
7.1	Basic Map-Reduce .....	68
7.2	Partitioning and Combining .....	69
7.3	Composing Map-Reduce Calculations .....	72
7.3.1	<i>A Two Stage Map-Reduce Example</i> .....	73
7.3.2	<i>Incremental Map-Reduce</i> .....	76
7.4	Further Reading .....	77
7.5	Key Points .....	77
Part II:	Implement .....	79
Chapter 8:	Key-Value Databases .....	81
8.1	What Is a Key-Value Store .....	81
8.2	Key-Value Store Features .....	83

8.2.1	<i>Consistency</i> .....	83
8.2.2	<i>Transactions</i> .....	84
8.2.3	<i>Query Features</i> .....	84
8.2.4	<i>Structure of Data</i> .....	86
8.2.5	<i>Scaling</i> .....	86
<b>8.3</b>	<b>Suitable Use Cases</b> .....	<b>87</b>
8.3.1	<i>Storing Session Information</i> .....	87
8.3.2	<i>User Profiles, Preferences</i> .....	87
8.3.3	<i>Shopping Cart Data</i> .....	87
<b>8.4</b>	<b>When Not to Use</b> .....	<b>87</b>
8.4.1	<i>Relationships among Data</i> .....	87
8.4.2	<i>Multioperation Transactions</i> .....	88
8.4.3	<i>Query by Data</i> .....	88
8.4.4	<i>Operations by Sets</i> .....	88
<b>Chapter 9:</b>	<b>Document Databases</b> .....	<b>89</b>
<b>9.1</b>	<b>What Is a Document Database?</b> .....	<b>90</b>
<b>9.2</b>	<b>Features</b> .....	<b>91</b>
9.2.1	<i>Consistency</i> .....	91
9.2.2	<i>Transactions</i> .....	92
9.2.3	<i>Availability</i> .....	93
9.2.4	<i>Query Features</i> .....	94
9.2.5	<i>Scaling</i> .....	95
<b>9.3</b>	<b>Suitable Use Cases</b> .....	<b>97</b>
9.3.1	<i>Event Logging</i> .....	97
9.3.2	<i>Content Management Systems, Blogging Platforms</i> .....	98
9.3.3	<i>Web Analytics or Real-Time Analytics</i> .....	98
9.3.4	<i>E-Commerce Applications</i> .....	98
<b>9.4</b>	<b>When Not to Use</b> .....	<b>98</b>
9.4.1	<i>Complex Transactions Spanning Different Operations</i> .....	98
9.4.2	<i>Queries against Varying Aggregate Structure</i> .....	98
<b>Chapter 10:</b>	<b>Column-Family Stores</b> .....	<b>99</b>
<b>10.1</b>	<b>What Is a Column-Family Data Store?</b> .....	<b>99</b>
<b>10.2</b>	<b>Features</b> .....	<b>100</b>
10.2.1	<i>Consistency</i> .....	103
10.2.2	<i>Transactions</i> .....	104
10.2.3	<i>Availability</i> .....	104

10.2.4	<i>Query Features</i> .....	105
10.2.5	<i>Scaling</i> .....	107
10.3	<b>Suitable Use Cases</b> .....	107
10.3.1	<i>Event Logging</i> .....	107
10.3.2	<i>Content Management Systems, Blogging Platforms</i> .....	108
10.3.3	<i>Counters</i> .....	108
10.3.4	<i>Expiring Usage</i> .....	108
10.4	<b>When Not to Use</b> .....	109
Chapter 11:	<b>Graph Databases</b> .....	111
11.1	<b>What Is a Graph Database?</b> .....	111
11.2	<b>Features</b> .....	113
11.2.1	<i>Consistency</i> .....	114
11.2.2	<i>Transactions</i> .....	114
11.2.3	<i>Availability</i> .....	115
11.2.4	<i>Query Features</i> .....	115
11.2.5	<i>Scaling</i> .....	119
11.3	<b>Suitable Use Cases</b> .....	120
11.3.1	<i>Connected Data</i> .....	120
11.3.2	<i>Routing, Dispatch, and Location-Based Services</i> .....	120
11.3.3	<i>Recommendation Engines</i> .....	121
11.4	<b>When Not to Use</b> .....	121
Chapter 12:	<b>Schema Migrations</b> .....	123
12.1	<b>Schema Changes</b> .....	123
12.2	<b>Schema Changes in RDBMS</b> .....	123
12.2.1	<i>Migrations for Green Field Projects</i> .....	124
12.2.2	<i>Migrations in Legacy Projects</i> .....	126
12.3	<b>Schema Changes in a NoSQL Data Store</b> .....	128
12.3.1	<i>Incremental Migration</i> .....	130
12.3.2	<i>Migrations in Graph Databases</i> .....	131
12.3.3	<i>Changing Aggregate Structure</i> .....	132
12.4	<b>Further Reading</b> .....	132
12.5	<b>Key Points</b> .....	132
Chapter 13:	<b>Polyglot Persistence</b> .....	133
13.1	<b>Disparate Data Storage Needs</b> .....	133
13.2	<b>Polyglot Data Store Usage</b> .....	134
13.3	<b>Service Usage over Direct Data Store Usage</b> .....	136

13.4	Expanding for Better Functionality .....	136
13.5	Choosing the Right Technology .....	138
13.6	Enterprise Concerns with Polyglot Persistence .....	138
13.7	Deployment Complexity .....	139
13.8	Key Points .....	140
Chapter 14:	Beyond NoSQL .....	141
14.1	File Systems .....	141
14.2	Event Sourcing .....	142
14.3	Memory Image .....	144
14.4	Version Control .....	145
14.5	XML Databases .....	145
14.6	Object Databases .....	146
14.7	Key Points .....	146
Chapter 15:	Choosing Your Database .....	147
15.1	Programmer Productivity .....	147
15.2	Data-Access Performance .....	149
15.3	Sticking with the Default .....	150
15.4	Hedging Your Bets .....	150
15.5	Key Points .....	151
15.6	Final Thoughts .....	152
Bibliography	.....	153
Index	.....	157

*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 <i>Riak</i>
Document (“Document Databases,” p. 89)	CouchDB <i>MongoDB</i> OrientDB RavenDB Terrastore
Column-Family (“Column-Family Stores,” p. 99)	Amazon SimpleDB <i>Cassandra</i> HBase Hypertable
Graph (“Graph Databases,” p. 111)	FlockDB HyperGraphDB Infinite Graph <i>Neo4J</i> 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

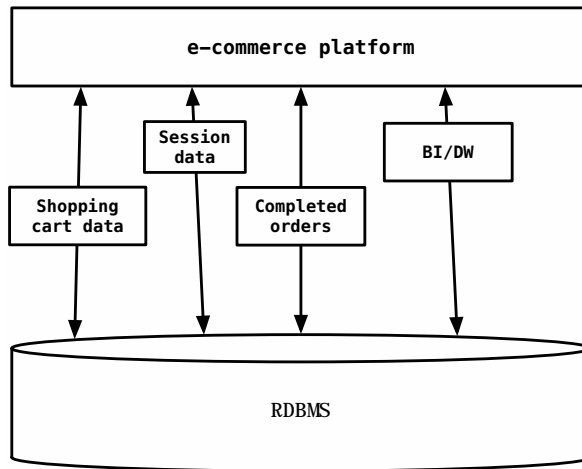


Figure 13.1 Use of RDBMS for every aspect of storage for the application

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

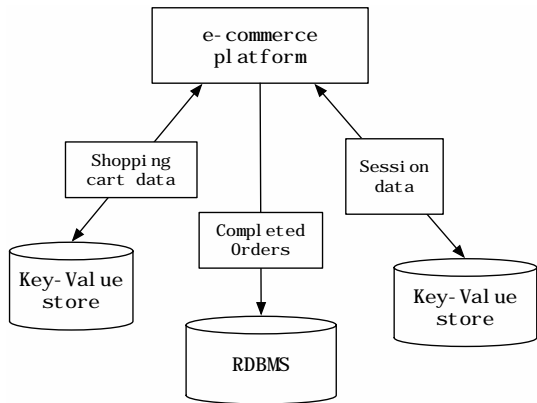


Figure 13.2 Use of key-value stores to offload session and shopping cart data storage

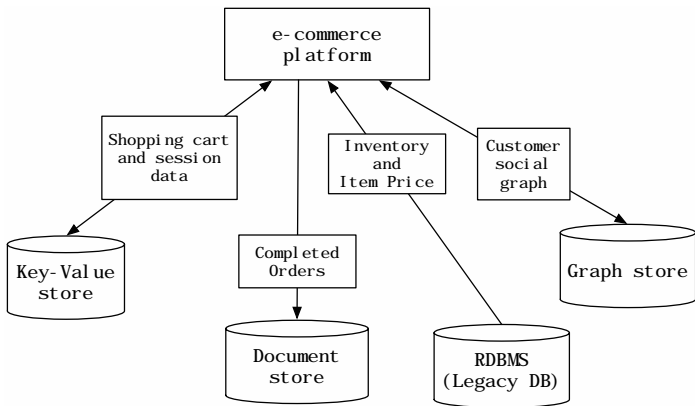


Figure 13.3 Example implementation of polyglot persistence

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 single 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.

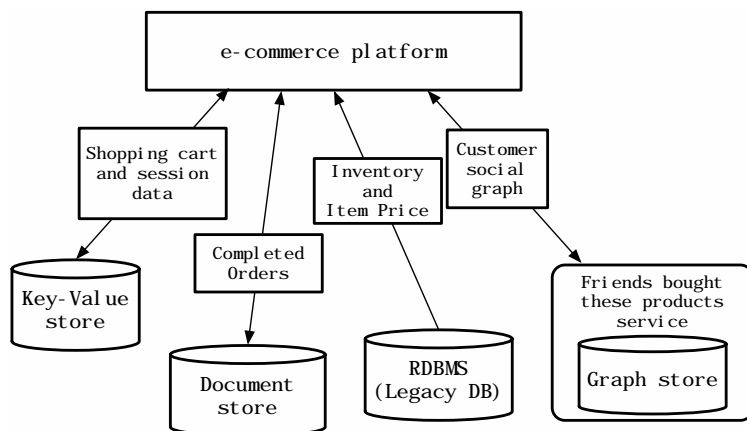


Figure 13.4 Example implementation of wrapping data stores into services

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

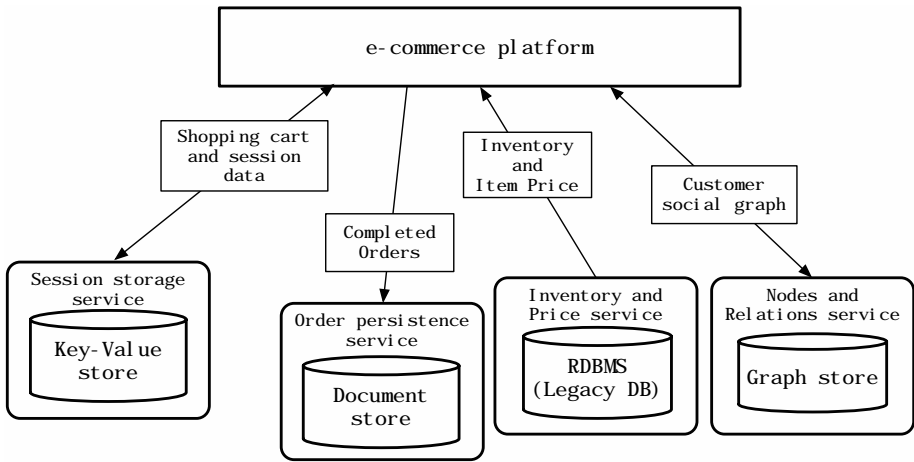
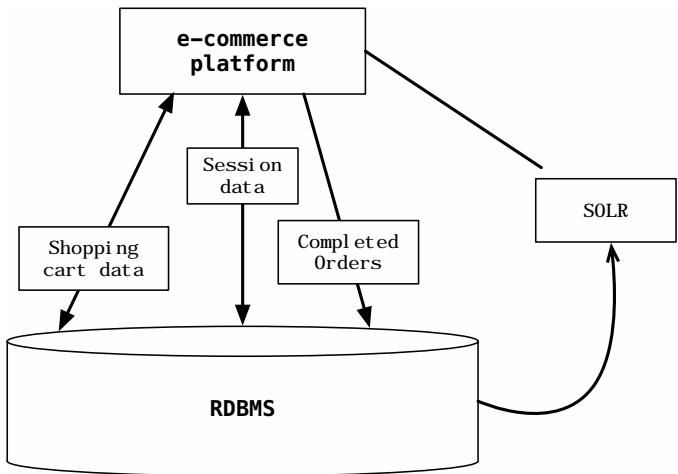


Figure 13.5 Using services instead of talking to databases

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

ACID (Atomic, Consistent, Isolated, and Durable) transactions, 19

in column-family databases, 109

in graph databases, 28, 50, 114–115

in relational databases, 10, 26

vs. BASE, 56

ad banners, 108–109

aggregate-oriented databases, 14, 19–23, 147

atomic updates in, 50, 61

disadvantages of, 30

no ACID transactions in, 50

performance of, 149

vs. graph databases, 28

aggregates, 14–23

changing structure of, 98, 132

modeling, 31

real-time analytics with, 33

updating, 26

agile methods, 123

Amazon, 9

*See also* DynamoDB, SimpleDB

analytics

counting website visitors for, 108

of historic information, 144

real-time, 33, 98

Apache Pig language, 76

Apache ZooKeeper library, 104, 115

application databases, 7, 146

updating materialized views in, 31

arcs (graph databases). *See* edges

atomic cross-document operations, 98

atomic rebalancing, 58

atomic transactions, 92, 104

atomic updates, 50, 61

automated failovers, 94

automated merges, 48

automated rollbacks, 145

auto-sharding, 39

availability, 53

in column-family databases, 104–105

in document databases, 93

in graph databases, 115

vs. consistency, 54

*See also* CAP theorem

- CAP (Consistency, Availability, and Partition tolerance) theorem, 53–56
    - for document databases, 93
    - for Riak, 86
  - CAS (compare-and-set) operations, 62
  - Cassandra DB, 10, 21–22, 99–109
    - availability in, 104–105
    - column families in:
      - commands for, 105–106
      - standard, 101
      - super, 101–102
    - columns in, 100
      - expiring, 108–109
      - indexing, 106–107
      - reading, 107
      - super, 101
    - compaction in, 103
    - consistency in, 103–104
    - ETL tools for, 139
    - hinted handoff in, 104
    - keyspaces in, 102–104
    - memtables in, 103
    - queries in, 105–107
    - repairs in, 103–104
    - replication factor in, 103
    - scaling in, 107
    - SSTables in, 103
    - timestamps in, 100
    - transactions in, 104
    - wide/skinny rows in, 23
  - clients, processing on, 67
  - Clojure language, 145
  - cloud computing, 149
  - clumping, 39
  - clusters, 8–10, 67–72, 76, 149
    - in file systems, 8
    - in Riak, 87
    - resiliency of, 8
  - column-family databases, 21–23, 99–109
    - ACID transactions in, 109
    - columns for materialized views in, 31
    - combining peer-to-peer replication and sharding in, 43–44
    - consistency in, 103–104
    - modeling for, 34
    - performance in, 103
    - schemalessness of, 28
    - vs. key-value databases, 21
    - wide/skinny rows in, 23
  - combinable reducers, 70–71
  - compaction (Cassandra), 103
  - compatibility, backward, 126, 131
  - concurrency, 145
    - in file systems, 141
    - in relational databases, 4
    - offline, 62
  - conditional updates, 48, 62–63
  - conflicts
    - key, 82
    - read-write, 49–50
    - resolving, 64
    - write-write, 47–48, 64
  - consistency, 47–59
    - eventual, 50, 84
    - in column-family databases, 103–104
    - in graph databases, 114
    - in master-slave replication, 52
    - in MongoDB, 91
    - logical, 50
    - optimistic/pessimistic, 48
    - read, 49–52, 56
    - read-your-writes, 52
    - relaxing, 52–56
    - replication, 50
    - session, 52, 63
    - trading off, 57
    - update, 47, 56, 61
    - vs. availability, 54
    - write, 92
    - See also* CAP theorem
  - content hashes, 62–63
  - content management systems, 98, 108
  - CouchDB, 10, 91
    - conditional updates in, 63
    - replica sets in, 94
  - counters, for version stamps, 62–63
  - CQL (Cassandra Query Language), 10, 106
  - CQRS (Command Query Responsibility Segregation), 143
  - cross-document operations, 98
  - C-Store DB, 21
  - Cypher language, 115–119
- ## D
- Data Mapper and Repository pattern, 151
  - data models, 13, 25
    - aggregate-oriented, 14–23, 30
    - document, 20
    - key-value, 20
    - relational, 13–14



- data redundancy, 94
  - databases
    - choosing, 7, 147–152
    - deploying, 139
    - encapsulating in explicit layer, 141
    - NoSQL, definition of, 10–11
    - shared integration of, 4, 6
  - Datastax Ops Center, 139
  - DBDeploy framework, 125
  - DBMaintain tool, 126
  - deadlocks, 48
  - demo access, 108
  - Dependency Network pattern, 77
  - deployment complexity, 139
  - Dijkstra’s algorithm, 118
  - disaster recovery, 94
  - distributed file systems, 76, 141
  - distributed version control systems, 48
    - version stamps in, 64
  - distribution models, 37–43
    - See also* replications, sharding, single server approach
  - document databases, 20, 23, 89–98
    - availability in, 93
    - embedding child documents into, 90
    - indexes in, 25
    - master-slave replication in, 93
    - performance in, 91
    - queries in, 25, 94–95
    - replica sets in, 94
    - scaling in, 95
    - schemalessness of, 28, 98
    - XML support in, 146
  - domain buckets (Riak), 83
  - Domain-Driven Design, 14
  - DTDs (Document Type Definitions), 146
  - durability, 56–57
  - DynamoDB, 9, 81, 100
    - shopping carts in, 55
  - Dynomite DB, 10
- E**
- early prototypes, 109
  - e-commerce
    - data modeling for, 14
    - flexible schemas for, 98
    - polyglot persistence of, 133–138
    - shopping carts in, 55, 85, 87
  - edges (graph databases), 26, 111
  - eligibility rules, 26
  - enterprises
    - commercial support of NoSQL for, 138–139
    - concurrency in, 4
    - DB as backing store for, 4
    - event logging in, 97
    - integration in, 4
    - polyglot persistence in, 138–139
    - security of data in, 139
  - error handling, 4, 145
  - etags, 62
  - ETL tools, 139
  - Evans, Eric, 10
  - event logging, 97, 107–108
  - event sourcing, 138, 142, 144
  - eventual consistency, 50
    - in Riak, 84
  - expiring usage, 108–109
- F**
- failovers, automated, 94
  - file systems, 141
    - as backing store for RDBMS, 3
    - cluster-aware, 8
    - concurrency in, 141
    - distributed, 76, 141
    - performance of, 141
    - queries in, 141
  - FlockDB, 113
    - data model of, 27
    - node distribution in, 115
- G**
- Gilbert, Seth, 53
  - Google, 9
    - Google BigTable. *See* BigTable
    - Google File System, 141
  - graph databases, 26–28, 111–121, 148
    - ACID transactions in, 28, 50, 114–115
    - aggregate-ignorance of, 19
    - availability in, 115
    - consistency in, 114
    - creating, 113
    - edges (arcs) in, 26, 111
    - held entirely in memory, 119
    - master-slave replication in, 115
    - migrations in, 131
    - modeling for, 35
    - nodes in, 26, 111–117
    - performance of, 149

graph databases (*continued*)  
 properties in, 111  
 queries in, 115–119  
 relationships in, 111–121  
 scaling in, 119  
 schemalessness of, 28  
 single server configuration of, 38  
 traversing, 111–117  
 vs. aggregate databases, 28  
 vs. relational databases, 27, 112  
 wrapping into service, 136  
 Gremlin language, 115  
 GUID (Globally Unique Identifier), 62

## H

Hadoop project, 67, 76, 141  
 HamsterDB, 81  
 hash tables, 62–63, 81  
 HBase DB, 10, 21–22, 99–100  
 Hector client, 105  
 Hibernate framework, 5, 147  
 hinted handoff, 104  
 hive DB, 76  
 hot backup, 40, 42  
 hotel booking, 4, 55  
 HTTP (Hypertext Transfer Protocol), 7  
   interfaces based on, 85  
   updating with, 62  
 Hypertable DB, 10, 99–100

## I

iBATIS, 5, 147  
 impedance mismatch, 5, 12  
 inconsistency  
   in shopping carts, 55  
   of reads, 49  
   of updates, 56  
   window of, 50–51, 56  
 indexes  
   bit-mapped, 106  
   in document databases, 25  
   stale data in, 138  
   updating, 138  
 Infinite Graph DB, 113  
   data model of, 27  
   node distribution in, 114–115  
 initial tech spikes, 109  
 integration databases, 6, 11  
 interoperability, 7

## J

JSON (JavaScript Object Notation), 7,  
 94–95, 146

## K

keys (key-value databases)  
 composite, 74  
 conflicts of, 82  
 designing, 85  
 expiring, 85  
 grouping into partitions, 70  
 keyspaces (Cassandra), 102–104  
 key-value databases, 20, 23, 81–88  
   consistency of, 83–84  
   modeling for, 31–33  
   no multiple key operations in, 88  
   schemalessness of, 28  
   sharding in, 86  
   structure of values in, 86  
   transactions in, 84, 88  
   vs. column-family databases, 21  
   XML support in, 146

## L

Liquibase tool, 126  
 location-based services, 120  
 locks  
   dead, 48  
   offline, 52  
 lost updates, 47  
 Lotus DB, 91  
 Lucene library, 85, 88, 116  
 Lynch, Nancy, 53

## M

MapReduce framework, 67  
 map-reduce pattern, 67–77  
   calculations with, 72  
   incremental, 31, 76–77  
   maps in, 68  
   materialized views in, 76  
   partitions in, 70  
   reusing intermediate outputs in, 76  
   stages for, 73–76  
 master-slave replication, 40–42  
   appointing masters in, 41, 57  
   combining with sharding, 43  
   consistency of, 52  
   in document databases, 93

- in graph databases, 115
- version stamps in, 63
- materialized views, 30
  - in map-reduce, 76
  - updating, 31
- Memcached DB, 81, 87
- memory images, 144–145
- memtables (Cassandra), 103
- merges, automated, 48
- migrations, 123–132
  - during development, 124, 126
  - in graph databases, 131
  - in legacy projects, 126–128
  - in object-oriented databases, 146
  - in schemaless databases, 128–132
  - incremental, 130
  - transition phase of, 126–128
- mobile apps, 131
- MongoDB, 10, 91–97
  - collections in, 91
  - consistency in, 91
  - databases in, 91
  - ETL tools for, 139
  - queries in, 94–95
  - replica sets in, 91, 93, 96
  - schema migrations in, 128–131
  - sharding in, 96
  - si aveOk parameter in, 91–92, 96
  - terminology in, 89
  - WriteConcern parameter in, 92
- MongoDB Monitoring Service, 139
- MyBatis Migrator tool, 126
- MySQL DB, 53, 119

## N

- Neo4J DB, 113–118
  - ACID transactions in, 114–115
  - availability in, 115
  - creating graphs in, 113
  - data model of, 27
  - replicated slaves in, 115
  - service wrapping in, 136
- nodes (graph databases), 26, 111
  - distributed storage for, 114
  - finding paths between, 117
  - indexing properties of, 115–116
- nonuniform data, 10, 28, 30
- NoSQL databases
  - advantages of, 12
  - definition of, 10–11

- lack of support for transactions in, 10, 61
- running of clusters, 10
- schemalessness of, 10

## O

- object-oriented databases, 5, 146
  - migrations in, 146
  - vs. relational databases, 6
- offline concurrency, 62
- offline locks, 52
- Optimistic Offline Lock, 62
- Oracle DB
  - redo log in, 104
  - terminology in, 81, 89
- Oracle RAC Server, 8
- OrientDB, 91, 113
- ORM (Object-Relational Mapping)
  - frameworks, 5–6, 147
- Oskarsson, Johan, 9

## P

- partition tolerance, 53–54
  - See also* CAP theorem
- partitioning, 69–70
- peer-to-peer replication, 42–43
  - durability of, 58
  - inconsistency of, 43
  - version stamps in, 63–64
- Pentaho tool, 139
- performance
  - and sharding, 39
  - and transactions, 53
  - binary protocols for, 7
  - caching for, 39, 137
  - data-access, 149–150
  - in aggregate-oriented databases, 149
  - in column-family databases, 103
  - in document databases, 91
  - in graph databases, 149
  - responsiveness of, 48
  - tests for, 149
- pipes-and-filters approach, 73
- polyglot persistence, 11, 133–139, 148
  - and deployment complexity, 139
  - in enterprises, 138–139
- polyglot programming, 133–134
- processing, on clients/servers, 67
- programmer productivity, 147–149
- purchase orders, 25

## Q

## queries

- against varying aggregate structure, 98
  - by data, 88, 94
  - by key, 84–86
  - for files, 141
  - in column-family databases, 105–107
  - in document databases, 25, 94–95
  - in graph databases, 115–119
  - precomputed and cached, 31
  - via views, 94
- quorums, 57, 59
- read, 58
  - write, 58, 84

## R

Rails Active Record framework, 147

RavenDB, 91

- atomic cross-document operations in, 98
- replica sets in, 94
- transactions in, 92

RDBMS. *See* relational databases

## reads

- consistency of, 49–52, 56, 58
- horizontal scaling for, 94, 96
- inconsistent, 49
- multiple nodes for, 143
- performance of, 52
- quorums of, 58
- repairs of, 103
- resilience of, 40–41
- separating from writes, 41
- stale, 56

read-write conflicts, 49–50

read-your-writes consistency, 52

Real Time Analytics, 33

Real Time BI, 33

rebalancing, atomic, 58

recommendation engines, 26, 35, 121, 138

Redis DB, 81–83

redo log, 104

reduce functions, 69

- combinable, 70–71

regions. *See* map-reduce pattern, partitions in

Rekon browser for Riak, 139

relational databases (RDBMS), 13, 17

- advantages of, 3–5, 7–8, 150
- aggregate-ignorance of, 19
- backing store in, 3
- clustered, 8

columns in, 13, 90

concurrency in, 4

defining schemas for, 28

impedance mismatch in, 5, 12

licensing costs of, 8

main memory in, 3

modifying multiple records at once in, 26

partitions in, 96

persistence in, 3

relations (tables) in, 5, 13

schemas for, 29–30, 123–128

security in, 7

sharding in, 8

simplicity of relationships in, 112

strong consistency of, 47

terminology in, 81, 89

transactions in, 4, 26, 92

tuples (rows) in, 5, 13–14

views in, 30

vs. graph databases, 27, 112

vs. object-oriented databases, 6

XML support in, 146

relationships, 25, 111–121

dangling, 114

direction of, 113, 116, 118

in RDBMS, 112

properties of, 113–115

traversing, 111–117

RelaxNG, 146

replica sets, 91, 93, 96

replication factor, 58

in column-family databases, 103

in Riak, 84

replications, 37

combining with sharding, 43

consistency of, 42, 50

durability of, 57

over clusters, 149

performance of, 39

version stamps in, 63–64

*See also* master-slave replication,  
peer-to-peer replication

resilience

and sharding, 39

read, 40–41

responsiveness, 48

Riak DB, 81–83

clusters in, 87

controlling CAP in, 86

eventual consistency in, 84

HTTP-based interface of, 85

- link-walking in, 25
  - partial retrieval in, 25
  - replication factor in, 84
  - service wrapping in, 136
  - terminology in, 81
  - transactions in, 84
  - write tolerance of, 84
  - Riak Search, 85, 88
  - rich domain model, 113
  - rollbacks, automated, 145
  - routing, 120
  - rows (RDBMS). *See* tuples
- ## S
- scaffolding code, 126
  - scaling, 95
    - horizontal, 149
      - for reads, 94, 96
      - for writes, 96
    - in column-family databases, 107
    - in document databases, 95
    - in graph databases, 119
    - vertical, 8
  - Scatter-Gather pattern, 67
  - schemaless databases, 28–30, 148
    - implicit schema of, 29
    - schema changes in, 128–132
  - schemas
    - backward compatibility of, 126, 131
    - changing, 128–132
    - during development, 124, 126
    - implicit, 29
    - migrations of, 123–132
  - search engines, 138
  - security, 139
  - servers
    - maintenance of, 94
    - processing on, 67
  - service-oriented architecture, 7
  - services, 136
    - and security, 139
    - decomposing database layer into, 151
    - decoupling between databases and, 7
    - over HTTP, 7
  - sessions
    - affinity, 52
    - consistency of, 52, 63
    - expire keys for, 85
    - management of, 133
    - sticky, 52
    - storing, 57, 87
  - sharding, 37–38, 40, 149
    - and performance, 39
    - and resilience, 39
    - auto, 39
    - by customer location, 97
    - combining with replication, 43
    - in key-value databases, 86
    - in MongoDB, 96
    - in relational databases, 8
  - shared database integration, 4, 6
  - shopping carts
    - expire keys for, 85
    - inconsistency in, 55
    - persistence of, 133
    - storing, 87
  - shuffling, 70
  - SimpleDB, 99
    - inconsistency window of, 50
  - single server approach, 37–38
    - consistency of, 53
    - no partition tolerance in, 54
    - transactions in, 53
    - version stamps in, 63
  - single-threaded event processors, 145
  - snapshots, 142–143
  - social networks, 26, 120
    - relationships between nodes in, 117
  - Solr indexing engine, 88, 137, 141
  - split brain situation, 53
  - SQL (Structured Query Language), 5
  - SSTables (Cassandra), 103
  - stale data
    - in cache, 50
    - in indexes/search engines, 138
    - reading, 56
  - standard column families (Cassandra), 101
  - sticky sessions, 52
  - storage models, 13
  - Strozzi, Carlo, 9
  - super column families (Cassandra), 101–102
  - super columns (Cassandra), 101
  - system transactions, 61
- ## T
- tables. *See* relational databases, relations in
  - telemetric data from physical devices, 57
  - Terrastore DB, 91, 94
  - timestamps
    - consistent notion of time for, 64
    - in column-family databases, 100
    - of last update, 63

