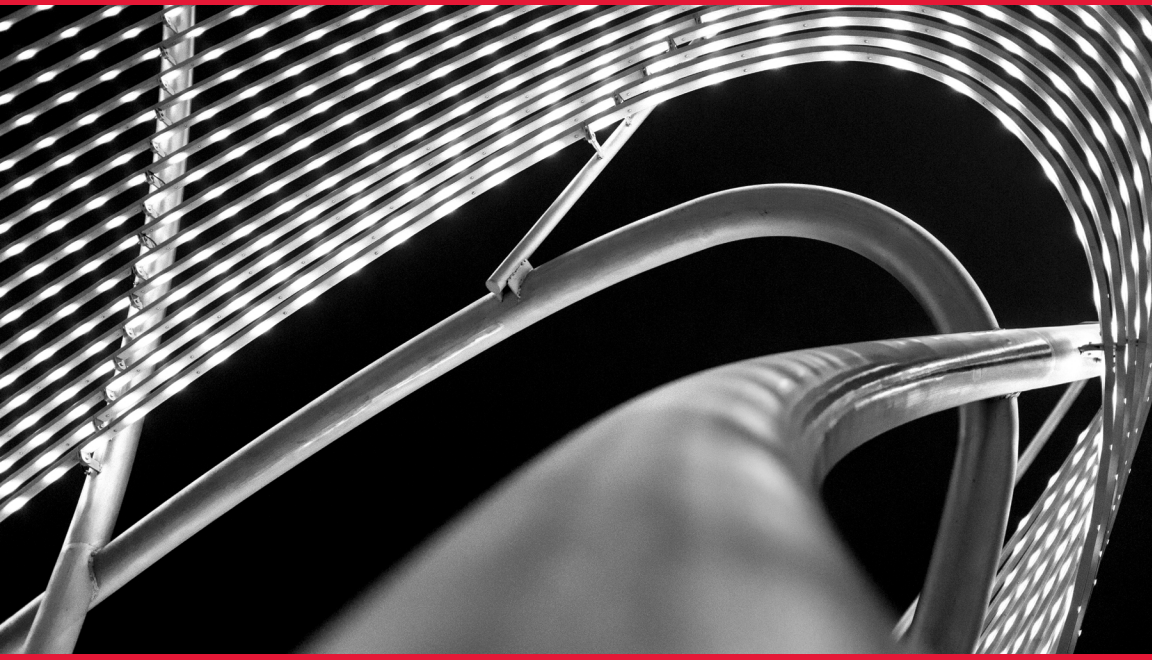
O'REILLY®

Compliments of MarkLogic®

Defining Data-Driven Software Development

Developing on a Multi-Model Database Management System



Eric Laquer

Your data deserves better.

There's nothing wrong with your data that a better database can't fix.

- Rely on a 100% trusted, enterprise-grade multi-model database
- Load your data *as-is*, no upfront data modeling required
- Unify your structured and unstructured data
- Build and deploy your apps in days, not months

www.marklogic.com

MarkLogic®

Defining Data-Driven Software Development

Developing on a Multi-Model Database Management System

Eric Laquer

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Defining Data-Driven Software Development
by Eric Laquer

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Shannon Cutt
Production Editor: Melanie Yarbrough
Copyeditor: Jasmine Kwityn
Proofreader: Eliahu Sussman

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

February 2017: First Edition

Revision History for the First Edition
2017-02-28: First Release

978-1-491-97925-9

[LSI]

Table of Contents

Preface

About This Report

Developers are urged every year to learn new frameworks and languages not only to enhance organizational productivity but also to stay relevant in an ever-changing industry. There's always a chance, however, that the tools you do decide to learn won't be around next year. And worse than wasting your time, you wasted the opportunity to learn something more relevant.

As you observe the staying power of NoSQL—and indeed the more mature multi-model database management system—it is natural to wonder: "Should I invest the time to learn yet another technology?"

This report aims to explore both the personal and professional benefits of developing on a multi-model database. As a developer with —ahem—a few decades of experience under my belt, I approached this exercise by looking at the entire development team. I evaluated how a multi-model approach would impact current roles as teams shift from rigid roles in a single-model environment to more fluid ones in a multi-model environment. The end result? We, as developers, move closer to the data becoming its own steward; it's no longer necessary to stand in long queues waiting for data to be readied. Instead, we can work directly with the data in the same useful forms in which it arrives from its sources and is delivered to its users.

With an understanding of the different ways of modeling and indexing data available to users of a multi-model database, it's possible to engage in data-driven development, because you are able to describe and quickly implement solutions for your customers. The tools and techniques described here leverage what you probably already know

about modeling and indexing of data. These new skills add the techniques required to move past the limitations of rows and tables altogether. Within this new paradigm, you and your team are given the ability to communicate—often on the fly—what a potential solution will look like for a customer. Agility is achieved by eliminating the time and effort required to tear the conceptual model down into tables, to build the conceptual model up from tables, and then to get each and every table definition approved through a separate DBA organization. Your efforts are applied directly to the actual data coming from and being sent to users in its native form of text, JSON, XML, and binary records and documents. The conceptual models that represent the value of your solution exist directly in the database tool immediately and interactively available for you, your team, and your stakeholders.

In the past, software developers might only have used specialized data modeling approaches at particular stages of development. For example, an entity-relation graph might have been used prior to creating a relational data schema. Recently, it has become more common for teams to assemble applications from data management tools specialized in dealing with one model type or another—for example, a graph database might be included to manage dynamic graph data on an ongoing basis. Multi-model technology accepts that our data is really many things. Working with one tool that provides different options for data modeling is the logical next step. Having a variety of representational forms and indexing techniques available simultaneously enables more creative and effective approaches to problems than can be found in the confining, single-schema world of tables, rows, and columns.

So do I think it's worth your time to invest in learning multi-model technology? Absolutely. If you are looking for a way to extend your career long beyond the next framework, you need to learn to work with all of your data. A model of all of the data flowing into and out of our systems is simply richer than the relational model—and the relational databases that implement that model—can handle. The limits of relational models are obvious when one looks at the bespoke integrations of additional separate tools that organizations have attached to their core data management systems in order to provide what their applications require. The need that many organizations have to use separate search, graph, and document tools suggests that data is naturally modeled in a variety of ways. I have

chosen to embrace multi-model tools that directly provide the modeling and indexing needed to create value.

In writing this report I looked at the advancements of DevOps and applied it to the extension of management of data as well. It's at this sweet spot that developers finally will have unlocked their true value to their organizations.

Major Points

- Organizations that can make high-quality data ubiquitously and securely available can focus on data-driven development—immediately answering the needs of their stakeholders.
- Relational database management systems (RDBMSs) managed within siloed organizations cannot effectively create many applications demanded by organizations today.
- Modern applications must process data that includes records, documents, videos, text, and semantic triples in a variety of formats, including XML, JSON, text, and binary.
- Organizations have concerns regarding data security, data privacy, data retention, data location, data availability, data protection in the face of a disaster, data search accuracy, and the analytic value of data.
- Applications that support both transactional and analytic use cases across huge heterogenous datasets are especially likely to break the RDBMS/siloed organization model.
- A new data-driven approach to software development and the next-generation multi-model database management systems have proven to be effective at creating applications where RDBMSs and siloed organizations have failed.
- Specialized software tools such as full-text search engines, single-model NoSQL databases, in-memory databases, and analytic databases cannot individually deliver a solution to core data management challenges.
- Compositions of multiple, specialized tools within a single application introduce additional integration complexity and risk.
- Software development teams should learn data-driven approaches and should develop multi-model database manage-

ment systems technology skills to respond to application demands that do not fit the classic RDBMS and siloed organizational approach.

- Using multi-model database management system tools in the context of a data-driven development team can transform the process of software development and improve the working lives of software developers.
- Aggregation of data into actionable knowledge is essential for organizations today, and data-driven software development using multi-model database technology is the best way to achieve that goal.

Acknowledgments

I'd like to thank the following people for making this report possible: Diane Burley, David Cassel, David Kaaret, Evelyn Kent, Dan McCreary, Rob Rudin, and Susan Anderson Laquer.

CHAPTER 1

Introduction

Data-Driven Approach

As software developers, we produce much of the intellectual capital that fuels economic and social change in today's digital world. The proliferation of data—and the rapid development of new tools to manage it—have put new demands on us. For 40 years, we put data into relational database management systems (RDBMSs)—we knew what the data looked like and database administrators (DBAs) knew what to do to store, secure, and retrieve that data. Now, with a variety of different data types and unprecedented data volumes, we are finding that as data becomes more complex and siloed in new purpose-built databases, we are being forced away from it. This creates ever-longer lags between specification and deployment.

Data-driven software development accepts the central role that data in its primary form takes in the applications that software developers create (Figure 1-1). Direct access to a multi-model database gives developers the ability to implement the transformations they need to accomplish. Documents are the heart and soul of information technology: two systems exchanging XML/JSON data, and a browser receiving an HTML/JSON response from a server or sending a JSON response back to it.

Outside of IT, much of what people work with is document-oriented as well. Everyday transactions such as sending emails, viewing pages in a web browser, filling out forms, creating Word documents, and doing financial modeling in Excel are all document-centric activities

that are not represented naturally in the restrictive relational data context. Multi-model databases accept data in the form the world already understands.

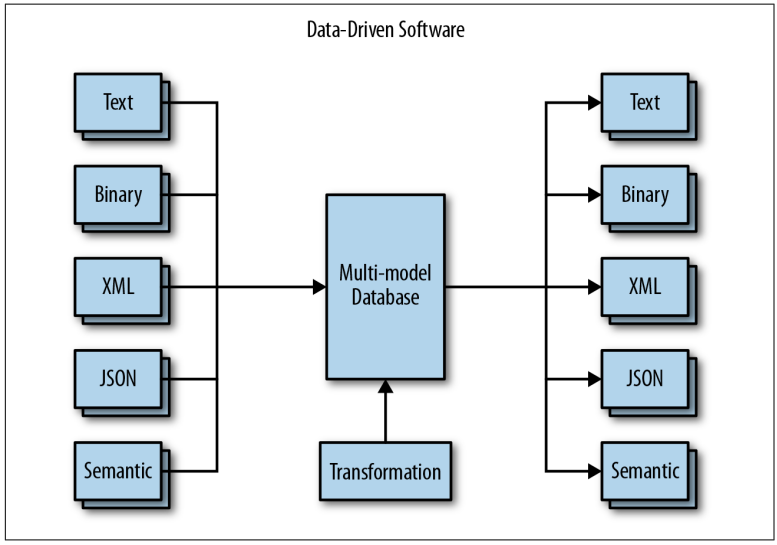


Figure 1-1. Data-driven software development

The data-driven approach requires a new way of thinking about the data; streamlining into a multi-model environment allows us to rapidly develop, providing newfound agility—and hopefully better job satisfaction.

Collection of Case Studies

As the following case studies and many other notable examples demonstrate, a data-driven approach and using a multi-model database enables business solutions that older approaches and older database technology simply could not accomplish within any tolerable time and budget constraints.

Fortune 50 Insurance Company Case Study

The first case study we'll look at involves a Fortune 50 insurance company that provides coverage for more than 20 million Americans. The company needed a better way to organize records for their roughly 25,000 employees.

Over the years, the insurance company had acquired a number of other firms, but its employee information remained in those firms' independent databases.

The company wanted to integrate its human resources data into a central repository so that it could be delivered to downstream systems for other real-time apps and batch analytics. The feeds included employee data, payroll data, employee evaluation data, and many other systems.

While much of the data was relational, integrating meant developing complex point-to-point data exchange solutions involving highly complex extract, transform, and load (ETL) processes that had to be written up front. This approach would be costly, time consuming, and perhaps impossible because of the rigid constraints of relational databases. It would also result in a brittle system that would require massive rework for any future changes.

Instead, the company created a flexible, document-based data model that could be developed in an Agile manner. This eliminated the need for a lot of complex ETL. The model was geared for communications between different formats, making it possible to decouple source data from target data, allowing for code reuse and making applications more resilient to change and growth in a number of data sources. The new system has ACID transactions along with full security, auditability, and data transparency.

The HR Data Hub successfully integrated more than 24 different HR data sources comprising about 140 data feeds. Today the data is consumed by more than 50 systems in over 200 different formats. The HR Data Hub handles terabytes of data and has a throughput of about 50 GB of data per day. The company delivered the solution in less than a year—a notable improvement over the five years it had previously been estimated to take.

Fortune 50 Bank Case Study

We'll now consider the example of a Fortune 500 bank that needed a better way to organize tens of thousands of IT assets, including everything from data centers, racks and servers, network routers, and applications.

Understanding the relationships between these assets and which individuals or services are impacted if they fail is critical to maintaining the bank's operations.

The data describing these different assets originates in a wide variety of data sources, and the bank needed to be able to perform analysis such as:

- Identifying affected software systems and business units when issues arise
- Cost reporting at a software system level
- Impact analysis and reporting for data center and infrastructure changes
- Effective, Google-like search across multiple silos
- Efficient data entry with reduced inconsistencies across various compliance initiatives
- Identification of gaps and redundancies across technology assets

A purely relational approach to handling these requirements would require extended development times and would provide poor performance—handling links between diverse assets is challenging when using a relational approach. As you may imagine, the data becomes hard to query through SQL, as there is no common data model linking the more-than-100 data sources that need to be integrated.

Instead of using a relational approach, the bank built a data repository and brought the data in as documents enriched with semantic triples (to describe links among assets). The result of this project is a technical asset inventory system that integrates all data silos and provides a real-time search and query interface to better analyze status, cost, impact analysis, gaps and redundancies, and much more.

Using document search and semantic queries along with SQL queries allowed the bank to greatly speed up development and improve capabilities over what would have been possible with a relational approach.

Additionally, the bank added a temporal layer to analyze events that occurred in the past and determine the impact of changes in its system.

Special-Purpose Solutions

The examples just discussed involved large and otherwise unmanageable situations. Sometimes we can find a *point* solution—or a special-purpose solution (or product)—that solves a specific problem rather than addressing all of the requirements that might otherwise be met with a multipurpose or multiservice product. Point solutions individually and collectively represent tremendous opportunities for improving application performance and for simplifying individual tasks. However, there is a danger that in rushing to solve one problem, we might create others. Here are some examples:

Symptom	Possible cure	Caution
Security audit reveals vulnerabilities across multiple tools storing data in multiple formats.	Work to implement multiple security measures across multiple tools and across multiple versions on an ongoing basis.	Full acceptance of the security demands of your application could have led to the selection of a multi-model database that directly addressed all the security requirements.
Queries are not effective because they are hard to compose, expensive to execute, and/or take too long to complete.	Add an analytical database and/or add a memory-based database.	Duplicating operational data can introduce inaccuracy and timeliness challenges.
Rigid schemas and "master data models" limit rate of change for large projects.	Add a new NoSQL database to allow storage of heterogeneous records.	Many features of traditional relational systems can be lost (e.g., ACID transaction support and record validation against schemas).
Semantic data does not fit well into traditional datastores.	Add a new triple datastore specialized for semantic data.	The triple datastore requires separate management, and synchronization can be complex. Also, the triple store will require its own security plan, implementation, maintenance, and audit.
Full-text search is required across both documents and operational records.	Add a full-text search engine and incorporate indexing with all record and document transactions.	Search makes all create, update, and delete (CRUD) operations more complex, while the accuracy of search results remains dependent on accurate synchronization. Also, the search tool will require its own security plan, implementation, maintenance, and audit.

Developing a Broader Perspective on Data Management

Developers can add value to their companies by focusing on delivering solutions faster and more securely than ever with a perspective that includes security, analysis, and operations concerns. Doing so will help them responsibly advance better approaches to data management.

As developers, we have seen this to some degree with the advancement of development operations (DevOps). DevOps is a collision of two disciplines of operations and development engineering that forged a new way of collaborating and problem solving. DevOps encouraged both camps to embrace some of the challenges previously encapsulated inside operations. In return for our investment in that, we have inherited many new processes and tools.

In a way, DevOps offers us a chance to frame our collective problems as collective opportunities:

"When your Ops folks start thinking about development and your Dev folks starting thinking about operations, you have a single team that works together with continuous development, continuous testing, and continuous deployments."[1]

This perspective, coupled with the use of a multi-model database, can dramatically transform how developers approach and solve everyday challenges, as we have seen in the case studies presented earlier. The following table illustrates other common problems that could benefit from a new approach:

Everyday problem	Better solution
A query across operational data cannot be run during the day because it would overload the servers.	With a multi-model database, specialized indexes combining full-text and traditional SQL concepts of entities can be built directly into the database, where indexes are constantly and consistently maintained. This enables fast and accurate search without extraordinary system loading.
The schema is not set up to do that.	With a multi-model database, multiple schemas can be used within the same collection. Collections with no schemas (NoSQL) are also possible.

1 New Relic website

Everyday problem	Better solution
We have no approved tools to store triples.	With a multi-model database, triples can be stored alongside the documents and records to which they refer.
We cannot do a keyword search on that database.	With a multi-model database, all parts of all documents and records can be referred to through a single, integrated index.
We sharded the NoSQL database last year, and we cannot change the key now.	With a multi-model database, there is no need to select a single shard key early in an application lifecycle.
The NoSQL database is not in our disaster recovery plan, so we lost it when we restored the system.	With a multi-model database, structured and unstructured data is maintained with a single set of high-availability (HA) and disaster recovery (DR) mechanisms.

With a clearer idea of the dramatic impact that our data management tools can have on the success or failure of our projects, let's take a closer look at what this data is in the real world.

CHAPTER 2
Data Is Knowledge

Data has a larger definition than ever before in history,[1] and it is exploding on an almost incomprehensible scale.[2] A Tesla automobile creates 10 GB of data per mile through its sensors,[3] for instance.

Knowledge is the aggregation of all this data. Resistance to the true nature of our data stands between us and the natural assimilation of document, sensor, semantic, geospatial, and binary data into information we can act on. With the right organizational and technical support, these data components can work together in a mutually supportive manner.

In the case of a Tesla automobile, one piece of sensor data might be needed for split-second, life-or-death decision making, and it might play a role in a machine-learning mission that will collect data from millions of cars driving over millions of miles, spanning decades. Furthermore, that one piece of data could well have legal consequences and decision-making processes around it, in addition to possibly having a moral facet. With this potential impact and longevity, we need to take data modeling seriously from the very start.

1 "Why Self-Driving Cars Must Be Programmed to Kill", MIT Technology Review.

2 Peter Levine, "The End of Cloud Computing", Andreessen Horowitz.

3 Fred Lambert, "A look at Tesla's new Autopilot hardware suite: 8 cameras, 1 radar, ultrasonics & new supercomputer", electrek.

Questions to Ask Before You Begin Data Modeling

- What form does the data take at its point of origin?
- If models are stored separately, how can we correlate them?
- How do you access the data from the models?
- Is there a common vocabulary across these models?

Further, the subtler aspect of data is that it's "out there"—everywhere, in the cloud, virtualized and on premises, segmented, warehoused, and in data lakes. Having data spread across systems and stores raises new questions and huge concerns about how to query it, access it, back it up, keep it safe, and more. In addition, stringent regulations and legal ramifications require the dev team to consider data governance and provenance.

Questions Developers Need to Consider

- Where and when was the data created?
- What confidence do we have that the data is correct in form and in content?
- What decisions need to be made based on that data and when do those decisions need to be made?
- Who owns the data?
- What regulations control the storage and communication of the data?
- How much is the data worth if it is kept?
- How much would it cost if the data was lost?

But what about the DBA? Isn't that the person who should be considering all that? In a multi–data-model world, it isn't one DBA—it is one for every model type. In development, this further slows down arduous processes and makes external deadlines more difficult to meet. Regulatory bodies, competitors, and internal stakeholders won't allow it.

There is another way.

Specialized Data Management

While the demands placed on IT departments have escalated in recent years, there has also been a renaissance in database technology. Data infrastructures that provide integrated access to these technologies are far better able to handle new business needs. The new tools include:

Documents
: There are many datasets that are difficult to force into a SQL format. Using this kind of data in a relational system can require extensive data modeling and ETL. To provide natural access to it, the database should natively support document data. Ideally, there should be SQL query access to document data as appropriate.

Semantics
: Semantic triples provide links between data elements, and between data and logical facts. By querying the triples—often in conjunction with other data—relationships between the data can be discovered that may be difficult to find in traditional databases.

Bitemporal
: Bitemporal functionality allows a database to preserve its past state over time and allows the database to be queried as it existed at any point in the past. Being able to query a database as it existed when a report was created can make it far easier to perform forensics and understand how the values in the report were generated.

Geospatial
: Queries and other analysis often includes geographical components.

Search
: When dealing with complex data, there are many situations where the ability to filter by free text search is a powerful approach to finding needed data. This is especially the case when search can be combined with other, more structured query approaches.

SQL

: Because so much data is currently stored in relational systems and accessed through SQL, it is essential to have SQL access to data.

The rise of the multi-model database can allow the development team to bring in these data types without the years of gymnastics typically associated with that much scale and variety. To see what this means for you and your team, let's seek insight from DBAs, system operators (SysOps), and other professionals to gain their perspectives.

CHAPTER 3

What Software Developers Want

To think about how a team can work together for a more cohesive view of data management, it's necessary to step back and look at what's important to the various players. Let's start with the developer's perspective.

As developers, we have a substantial impact on the way our users do business. All we have to do is keep up with a never-ending list of new requirements, bug fixes, and technical debt.[1] How fast we can respond to these needs dictates how fast others can get on with their jobs.

To address the stream of changes, we need to be flexible. The software we work with needs to be flexible, too.

Freedom from Rigid Schemas

We've come to realize that requirements are constantly evolving. This is natural—as our understanding of business needs change, our software must adapt to the current demands. Attempting to define a schema up front that will handle all of our data and meet all of our needs is time consuming and frustrating—and changing requirements will affect what needs to be stored, updated, searched, and queried.

1 Martin Fowler, "TechnicalDebt", *martinfowler.com*

We understand that having a schema isn't bad at all. There's value in knowing what your data looks like. The challenge presented by relational databases isn't that they require a schema—it's that they require *exactly one*.

One of the hurdles to being nimble is the extremely painful process of schema changes. Identifying the needed change is just the first step, to be followed by a migration plan, scripting the change, adapting the business layer code to the new schema, and working with the DBA to find a window where the change can be made.

We need an approach to managing data that gives us the access and flexibility necessary to address the constantly changing demands of the business.

Schemas rigidly constrain data that will be persisted into a particular table (Figure 3-1). Every row of the table must match this schema, and if the schema changes, every row of existing data must be reformed. Some other tools can use schemas to validate data but still allow the ingestion of invalid data. Where ingestion of huge scale and complexity is required, this can save on data reconciliation costs.[2]

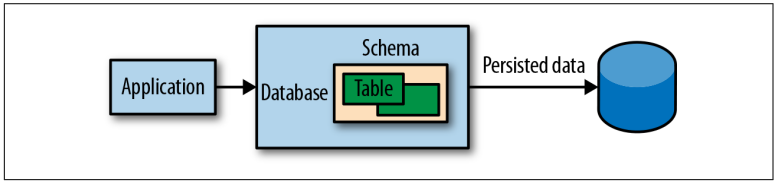


Figure 3-1. Rigid schema (RDBMS)

Dynamic Schemas

Without any schemas, almost any type of data can be persisted. If the data is Extensible Markup Language (XML), an explicit schema can exist on a per-document level. JavaScript Object Notation (JSON) documents can reference schemas as well using the JSON API or a competing standard.[3] XML and JSON are both considered

2 "Reconciling Your Enterprise Data Warehouse to Source Systems", presented by James Hanck, Santosh Pappachan, Scott Hertel, and Sunny Hiran.

3 JSON API: Specification for building APIs in JSON

self-describing[4] data. Text, comma-separated values (CSV), and other formats offer enough structure so that at least some record-based indexing can be done.

Selective Use of Schemas

Our applications deal with all types of collections, and schemas should be there to help us when and if we want them. When we are processing diverse types of data, we would like to be able to work without schemas, and when we want to maintain structure and rigor, we want schemas.

For example, some applications require only a Google-style free text search to find the data we require. At the other extreme, some applications require complex aggregates against rigidly fielded data types. We want tools that work with us as we accept, transform, and deliver data. By their nature, our applications make assumptions about the data they process. The old RDBMS model requires that homogeneous collections of records populate each table, and this assumption simply does not fit many of the situations we encounter. One can use a filesystem to persist heterogenous objects and one can use a directory to hold a collection. (See Figure 3-2.) Most of us have developed programs that use both an RDBMS and a filesystem. Such a system is using a polyglot persistence example (described in more detail momentarily). The point is simply that our applications require flexibility in how they persist data.

Multi-model databases provide a single system to store multiple types of data, including JSON, XML, text, CSV, triples (RDF), and binary. Accessing these data types from a single source using a common interface simplifies the application code, compared to using different systems to store the same information. This provides much greater flexibility to implement features that require data integration across these boundaries. These tools can therefore maintain homogeneous, constrained, and/or heterogeneous collections.

When such tools are used, applications can be written to be constrained "just enough." For example, saving to a homogeneous collection that is constrained to a single schema might be the right choice for storing financial data early in an application's lifecycle. A

4 The Self-Describing Web

constrained collection might be a good fit as that application evolves to handle more varied inputs. It might be enough if all the documents had just a few fields in common. Setting up a cross-reference sub-system in that same application might be best supported by a heterogeneous collection.

Taking this example a bit further, the cross-reference sub-system could benefit from full-text searches across all collections and data types. Such needs are satisfied in a polyglot persistence system with a dedicated search tool. Multi-model databases incorporate features that could support all aspects of data persistence for this hypothetical application.

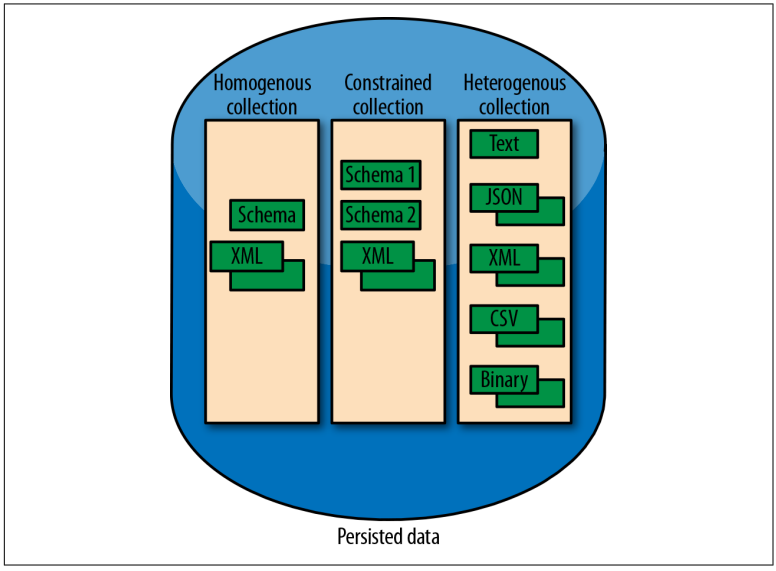


Figure 3-2. A multi-model DBMS should be capable of persisting homogenous, constrained, and heterogenous collections

Ability to Handle Different Data Types

The relational approach comes with a couple of assumptions: data comes in rows and columns, and the specific set of columns to be stored within a given table is the same for each row. Actual data, of course, comes in various shapes and sizes, and using tools that constrain our choices makes it more difficult for developers to meet the current and ongoing needs of our stakeholders. Persisted data across collections may include schema-full and schema-less data. Accessing

them through a single interface simplifies code and makes the delivery of value to our stakeholders more efficient.

Many software projects begin with existing sets of data. These may be from APIs offering JSON or XML, binary documents produced with desktop software, even linked data describing connections among entities. Applications may generate new types of data. There is no one-size-fits-all approach to storing information like this; the best tools will depend on what the data looks like and what one needs to do with it. For each of these types of data, an architect needs to determine the best way to store it.

Architectural Simplicity

As projects expand over time, more data and more types of data are included. If we are using a multi-model database, then our system can handle incoming data in its raw form and process it using tools appropriate to those types. A multi-model data management system that supports text, JSON, XML, CSV, triples, binary, and geospatial models—and also supports indexing across all of those models—can provide a robust basis for a variety of applications. Without multi-model data support, polyglot persistence solutions can be assembled. But that assembly can be difficult to get right.[5] As it turns out, performing join operations and building and invoking indexes for searching are best done by databases, not by application code.

Polyglot Persistence

Neal Ford coined the term *polyglot programming* to express the idea that 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.[6] This same concept can be applied to databases; you can have an application that talks to different databases using each for what they are best at to achieve an end goal, thus giving birth to *polyglot persistence*. While polyglot persistence has advantages, issues arise from needing to

5 Damon Feldman, "Avoiding the Franken-beast: Polyglot Persistence Done Right", MarkLogic.

6 Neal Ford, *The Productive Programmer* (O'Reilly)

integrate multiple tools to accomplish this. We characterize this as a *multiproduct* approach. (See Figure 3-3.)

As was mentioned before, to qualify as multiproduct, we need only two persistence mechanisms. Our simplest example illustrates the problem. If our application tier is writing to both a local filesystem and a database, the consistency of the records persisted on each is very different. If one has two instances of the application server running, files read or written to one server will not be the same as the files read and written from the other. Yes, this can be solved,[7] but this choice has made our lives and the lives of other professionals in our organization a little more complicated![8]

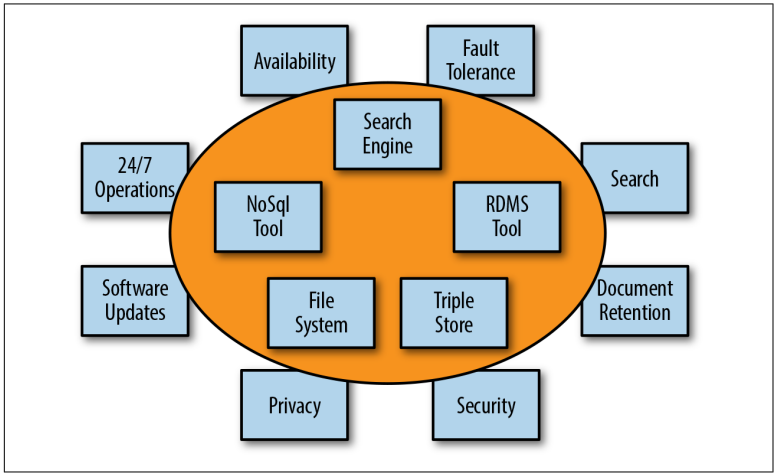


Figure 3-3. Multi-product, multi-model persistence

Multi-Model DBMS

With a unified multi-model database, all the issues surrounding persisting data in our organization still must be addressed—exactly once.

7 Red Hat Gluster Storage

8 Damon Feldman, "Avoiding the Franken-beast: Polyglot Persistence Done Right", MarkLogic.

Minimize Conversion

In an effort to maintain stability and to control complexity, many organizations will try to limit the types of data persistence they allow. This is quite understandable, but it trades one type of complexity for another. Rather than pushing our colleagues to adopt a myriad of persistence tools, we are asked to push and pull our data into the limited tools that are available. For us, this conversion code is some of the most complex, voluminous, and brittle code we write. Polyglot persistence and multi-model databases can both improve this particular situation.

Functionality

What will our applications do with the data? Our end users may want full-text searches, fast transactions, analytical data, or—most likely—a combination of many capabilities. However the solutions are structured, they would necessarily implement whichever set of capabilities is required. Even better, our solutions should be flexible enough to handle future needs that haven't surfaced yet. To maintain a smooth, Agile approach to feature delivery, having a fundamental set of tools available for all types of data likely to be encountered becomes a requirement (see Figure 3-4).

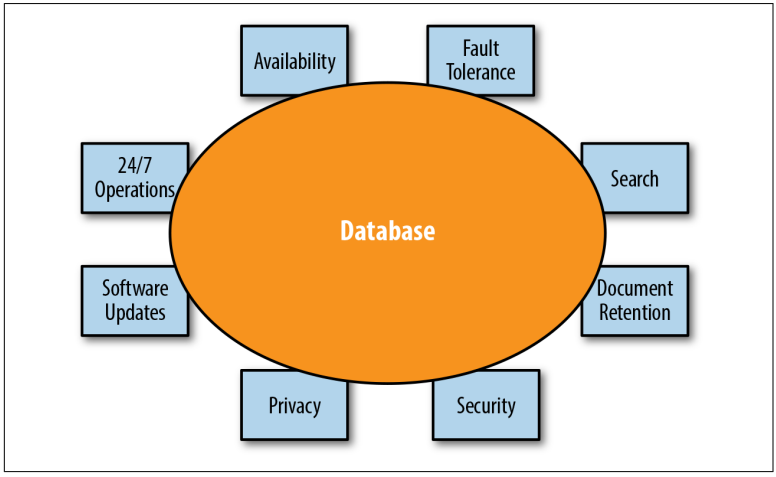


Figure 3-4. Characteristics of multi-model DBMS

Shaping Data and Impact on Other Roles

To address requirements in better and faster ways, we need to have access to the data, as well as the ability to shape it. We must be able to determine how to store a wide variety of data types, adjusting as we go.

Now that we have determined our "wants," let's look at the primary responsibility of others involved in the development process. By doing so, we can understand the impact of the changes we want to pursue on these other domains.

CHAPTER 4
What DBAs Want

DBAs maintain data in an RDBMS-centric information processing environment, and the typical DBA manages 40 databases containing an average of 5 TB of data.[1] Therefore, many have developed a comprehensive, if somewhat protective, view toward maintaining data integrity and accessibility. Perhaps as a consequence, there is some concern in the industry that DBAs are now becoming a limiting factor as organizations attempt to speed up their delivery pipeline.[2]

ACID Compliance

To ensure streams of data are *accurate* and *consistent*, ACID compliance might be a requirement placed on our database management system. To that end, let's review the ACID acronym:

Atomicity
: Make all changes or none.

Consistency
: All changes obey database rules.

Isolation
: Apply all changes step by step before showing any changes.

1 Noel Yuhanna, "How Many DBAs Do You Need To Support Databases?", Forrester blog.

2 Robert Reeves, "Is it time for an intervention on behalf of DBAs?", CIODIVE.

Durability
: Changes will be permanent.

Note: Some systems will claim "ACID" but ACID across one document is not the same as ACID across the dataset. Likewise, *eventual consistency* is not the same as *formal consistency*.

The choice of DBMS will determine whether the database can be trusted to provide the necessary reliability, or whether the developer will need to account for this in the application code.

Minimal Schema Changes

For DBAs, schema changes are critical transactions that need planning and careful execution. Frequent and reckless changes in schemas are both difficult to manage in RDBMSs and propagate risk in ways that are sometimes difficult to mitigate. Any NoSQL solution will offer ways to store documents with different schemas into the same table. These tools do not, however, mitigate issues raised by application linkage to particular data schemas. The key from the administrator's view is to ensure that the integrity of the data is not compromised in ways that adversely affect which data is delivered to end users.

High Availability

DBAs set up database software in such a way that the information will always be available to users. This can include configuration and maintenance of high-availability (HA) features, as well as disaster recovery (DR) features.

Fault Tolerance

Fault tolerance defines the ability of your application to maintain responsiveness in the face of infrastructure failures. This requires that a DBA replicate databases, and that facilities for "failing over" from one copy to another are available in whatever DBMS is selected.

Questions to Ask Moving Forward

If we are choosing to use a multi-model data management solution, our DBAs will want to know whether we have a standard operating procedure for dealing with existing data following each implicit or explicit schema change. Will your applications be able to handle the expected mixture of schemas at every juncture?

What about the availability for our application? Does the DBMS we are considering support HA, and what ongoing management will be required to maintain it? If our application must maintain some critical minimum level of continuous performance, does the DBMS support fault tolerance?

With so many concerns expressed here, one might legitimately ask how software developers can possibly address them all. The answers proposed here are organizational in nature. Developers might want to learn the specific features of the multi-model database products that are available, but that knowledge requires a collaborative approach from the organization implementing the solution to deliver the potential value of a data-driven approach.

To lay the groundwork for this type of cooperation, I'm going to systematically visit several areas of potential synergy. In many organizations, these functions are managed by separate groups and full adoption of a data-driven paradigm could involve transformation changes to the way organizations are, in fact, organized.

System administration, compliance and risk management, data analysis, business analysis, and the development process itself are addressed next.

CHAPTER 5

What the SysAdmin Wants

System operators (see Figure 5-1) are chartered with deploying and maintaining the compute, storage, and network resources required to support the users of our applications. The range of tools and techniques available to these professionals is enormous, but the most interesting trends for our purposes here are software-defined infrastructure (SDI) and the DevOps tool market.

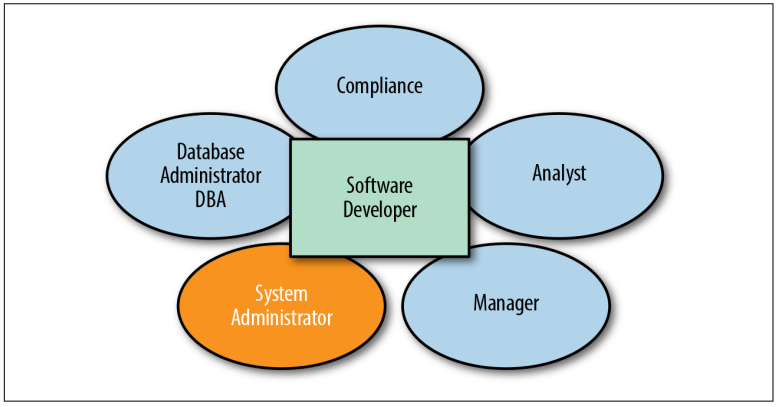


Figure 5-1. Software developer and the system administrator

Software-Defined Infrastructure

SDI performs the task of setting up individual hosts, clusters of hosts, and even global-scale service platforms with only a set of scripts. The technology enabling most of this power came with advances in cloud computing, but the market is now so dynamic

that tools and techniques exist for using SDI in cloud, on-premise, or hybrid deployment situations. These deployments can include components of every capacity and at every price point. In many cases, several types of storage are available, ranging from the very fast and expensive to the slow and inexpensive. Resources can be made available instantly for a few tenths of a second or might be reserved or purchased in anticipation of years of continuous operation.

DevOps and Serverless Architecture

System administrators are enabling DevOps to do much of the work done previously within the sysadmin domain. The combination of modern DevOps tools with SDI is a particularly potent way to expedite delivery value to stakeholders.

Teams can apply modern DevOps tools that largely automate this setup of services that support the needs of several stakeholders simultaneously. Security, privacy, audit, high availability, disaster recovery, and other features can be systematically built into platforms as they are created. Serverless architectures allow systems to exist in nascent forms that consume virtually no resources until service is required. Developers tracking these technologies will be able to deliver capabilities previously unachievable or impractical.

Storage Tiers

Does the application store offer easily segmentable tiers of data storage with varying requirements (i.e., fast for the newest and inexpensive for the archive)? If your data is sizable and your data management tools can direct the appropriate data to the right storage, you can save your organizations large sums over time by finding a tool that supports this and by helping configure the tool properly. Data stores that support this functionality save time and money compared to having to build such capabilities into application code.

Architectural Simplicity

The more components there are to a system, the more things can go wrong. Polyglot persistence has the advantage of providing the best tool for each part of the job, but also the disadvantage of being a

complex architecture. Each component of that architecture requires administrator skills, hardware resources, and maintenance. In addition, combining results from different storage engines needs to be done in application code, instead of concentrating data management in the database. Multi-model databases have the ability to simplify the architecture while still providing the broad range of capabilities required by modern systems.

Considering a Multi-Model DBMS

System operators pride themselves on deploying and maintaining robust systems. If an application calls for multiple models of data, we now have a decision to make. If we choose to deploy a polyglot storage system, we will need to give the deployment and maintenance aspects special consideration. We should at least consider a multi-model DBMS for the reasons just described.

Let's think about the process required to install and scale up or scale down your application. Could we hand the process to an experienced system administrator, and would they be able to complete the process without help from us or our team?

CHAPTER 6

Compliance—Risk Management Perspective

Governance, risk, and compliance (GRC) and enterprise risk management (ERM) professionals (see Figure 6-1) concern themselves with the things that could go terribly wrong. To prevent a disaster, like a leak of customers' private data, or the loss of customer confidence that could be caused by a denial-of-service attack, positive steps need to be taken. A more integrated relationship between information systems and risk involves both the real-time monitoring of situations, as well as the forensic re-creation of historic situations.

Data-driven developers need to be aware of these requirements and threats and be able to build architectures and deploy systems that meet these demands. Solutions to issues in other domains can impact compliance adversely. Security and privacy violations can be damaging whether they pass through online transaction processing (OLTP) or through online analytical processing (OLAP) processes (see Figure 6-2). Compliance needs to be universally applied whether working with a single multi-model database or with multiple persistence mechanisms in a polyglot persistence architecture.

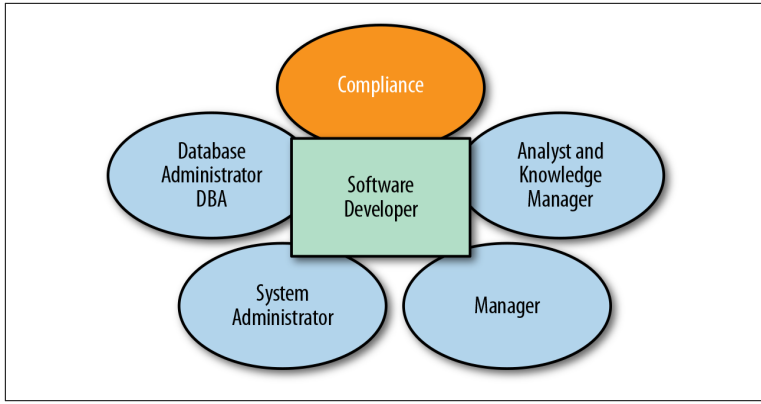


Figure 6-1. Software developer and compliance

Redaction and Field-Level Security

Compliance might impose on us seemingly contradictory requirements. For example, adherence to a regulation might require a field to be maintained, but the regulation might also state that it only be revealed to those holding special credentials. Our applications can synthesize such behaviors, but a data management system that supports specialized redaction and field-level security features to handle this directly could dramatically expedite our development of a compliant system, while simultaneously strengthening the compliance guarantees that our system can offer.

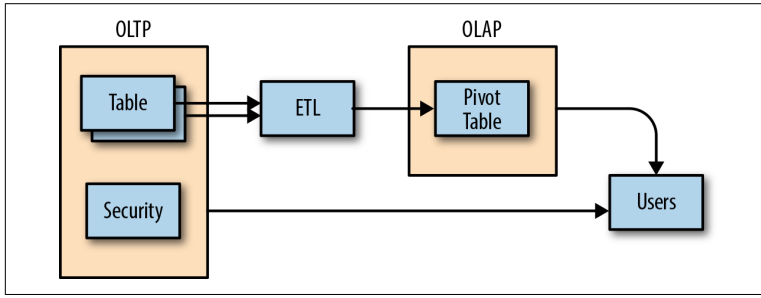


Figure 6-2. Data access through OTLP and OLAP

Network Security Transitioning to Data Security

With cloud deployments becoming more common and with rising awareness of threats from "insiders," the security of the filesystem on which your data management service stores your records no longer can be guaranteed in many situations. We are asked to get more involved in the physical aspects of our deployed systems to counter cyber security threats. Encryption-at-rest technology can help mitigate risks, as using this technology cuts off one of the major vectors of data piracy, which is the filesystem itself.

Taking into Account the Needs of a System

Our risk management and compliance teams might want us to select a data management system capable of granular control of data access down to the field level. They might also want our data management system to redact certain fields or sections of text based on the roles assigned to a particular reader. If we are working in a regulated industry, or if we think our industry might be regulated in the future, bitemporal data might be a "must have" to enable on-demand re-creation of our data's past states.

CHAPTER 7
What the Analyst Wants

Data-driven developers need to understand that data in their systems might have a lifecycle that extends far beyond the flow of transactions for which their specific application might be immediately responsible. Whether analysts are a part of your team or separated by several organizational divisions (Figure 7-1), the data they work with is the same data that powers the rest of the organization. Implementation artifacts can distort this, but the data characteristics important to those professions can and should be considered (see Figure 7-2).

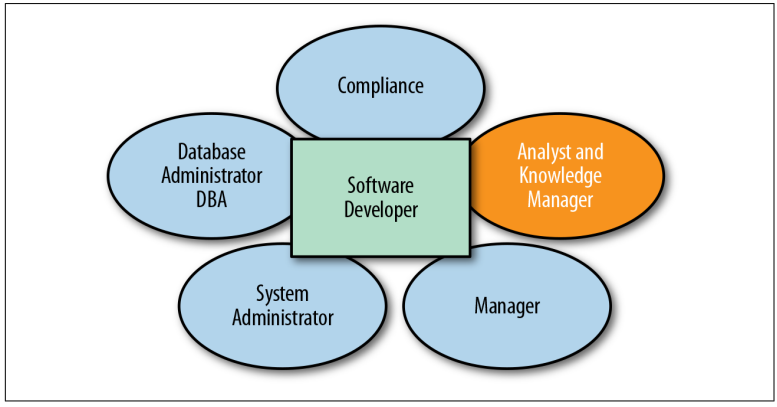


Figure 7-1. Software developer and analyst

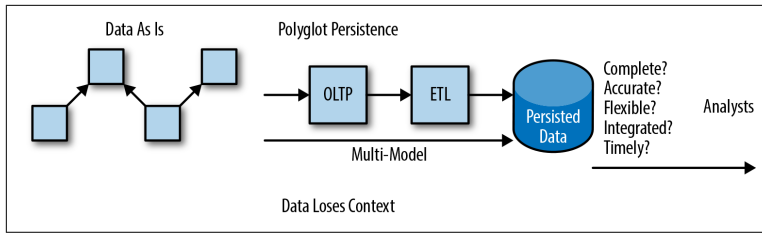


Figure 7-2. Data loses context

Knowledge management is even broader in scope. Introduction of this topic can lead many organizations to consider triple stores for the storage and retrieval of semantic data. Semantic technology is transformational in that it allows an organization to ask and answer questions effectively at a higher level. Ontological inquiries (both philosophical and information-science based) such as *Who are our real customers?* can create much more effective analytic answers to business questions such as *How do we increase our perceived value in the eyes of our customers?*

Once again, this describes the conditions needed for small, nimble groups to disrupt entire industries by asking a few of the right probing questions and by implementing solutions based on a better understanding of the real data. Developers who have not encountered requirements in this area yet might well encounter them soon.[1]

Extracting, Transforming, and Loading (ETL)

With RDBMS-centric applications, OLAP might well be extracting, transforming, and loading (ETL) records from their data stores into other stores specialized to do the sort of operations that analysts need to do.

ETL itself can be a formidable undertaking, but even with this effort, analytic results are delayed by at least a few hours. Small and nimble companies are disrupting markets by reacting faster than the established incumbent companies. Analytical flows that include ETL processes should be carefully considered. Also, ETL typically strips data

1 Abraham Bernstein, James Hendler, Natalya Noy, "A New Look at the Semantic Web", Communications of the ACM

from its source context and from its security protections. This might well complicate compliance efforts.

Complete and Integrated

Analysts want complete information about what is happening over a wide scope, which might go well beyond one siloed part of an organization. The less constrained the view, the better the analysis that can be done. What challenges data-driven developers, who must support analysts, is that data is spread out in most organizations across a myriad of siloed systems that do not have synchronized data models.

Collecting data with ETL is fraught with challenges. If our data management system can handle data "as is," then analysts can do their work more effectively.

Accurate

If accuracy is absolutely essential, as it is in financial institutions, then our OLTP systems' adherence to strict rules for updates need to be reinforced. ACID compliance has been a well-known feature of OLTP data systems historically. OLAP system were optimized for breadth of scope and flexibility over accuracy. If our analysis results require high accuracy, we might find that the classic ETL processes might not be up to the task. Architectural approaches like the construction of an Operational Data Hub could be called for.

Flexible and Timely

Flexible analysis support is necessary because the future is impossible to predict. OLAP systems are designed to support "index everything" approaches. The RDBMSs that support many OLTP data systems do not provide comprehensive indexes, but instead offer limited indexes that are purposefully built to speed the anticipated transactions. Unanticipated queries against such systems are not performant.

When all of the previous criteria are met, analysts have timely data with which to make decisions. Whether those decisions benefit the organization itself, or whether these outputs are sold to others, the value of timely analysis is far greater than analysis in retrospect.

Analysts will want to extract data from our systems in either batch or real-time modes. We should consider whether we want to prioritize analytics by adopting a data management system that can work with data "as is," without requiring uniform schemas across datasets.

CHAPTER 8

Focus on the Data

Meeting the "needs" and "wants" we've brought up requires two types of changes: new technologies and new ways of using them. Neither by itself will complete the change that developers need to accomplish.

Technology

Over the last several years, many new systems for storing, querying, and retrieving data have become available. We've looked at the various types of data storage systems that a project will have to deal with: document stores, graph databases, key/value stores, and other new types of databases that each bring unique capabilities to bear on the issues at hand. However, it is when these abilities are combined in a multi-model database that a development team can truly achieve data agility.

Multi-model databases allow flexible representation of a wide variety of data, with the details depending on which models the database supports. Once such a database is adopted, the team can develop expertise in that specific tool. By comparison, the polyglot persistence approach, in which several tools are combined to provide different capabilities, leads to developers becoming experts in a subset of those tools, making it harder to see how a change in one system will affect others.

Another benefit to combining storage in one multi-model system is the simplification of the application logic that interacts with it. If

multiple stores hold the application's data, then the application tier needs to synthesize any behaviors that cross those storage barriers. If one system holds the data in a format that isn't a natural representation for that data, then the application tier might need to reassemble the data back into a useful form.

The designers of an application also need to take into account that modern databases vary in more than just storage formats—capabilities that were once considered critical in a database are now left out of some. Choose a database that provides the functionality you need. If you need support for ACID transactions, don't try to implement that in your application layer; choose a database that provides that support. If you need zero-latency search after updates, make sure your database provides the search and query capability that you'll need. Let your database be the database; save your work in the application layer for your business logic.

Process

New databases won't solve our problems by themselves. If we take the same approach with them as we have taken with traditional databases, we will only get some of the benefit. How can we maximize our return?

When we choose a database that provides flexibility, we have to instill discipline in how we use that flexibility. If we decide that "anything goes," we'll break more than we fix.

The short answer is to take what we've learned from software engineering and DevOps and apply it to how we manage our data.

Skills

This report has covered a lot of ground. It has been suggested that we as software developers might need to accelerate our learning to better accommodate the needs of our stakeholders and to use the capabilities of our tools. Considering our perspective, we might well feel our rate of learning might have already maxed out.[1] The old adage to work smarter, not harder, applies here. By looking at a big-

1 Jay Xiong, *New Software Engineering Paradigm Based on Complexity Science: An Introduction to NSE*, *Springer Science & Business Media* (2011).

ger picture, we may well be able to identify a far smaller set of new processes and tools on which to focus our efforts than we had originally imagined, or feared.

With that said, to move forward in the accelerating software development world, better skills with whatever data management tools you select will be useful. Learning to utilize all of the facilities of a multi-model database could have extraordinary impact because whatever procedures you pass to the database will be done to whatever scale you might require. The simple API call you make might ultimately be run in parallel in 100 or more servers. As the "typical" server grows from 8 to 16 cores, for instance, your application could well double its performance if you have chosen correctly. A skillset along these lines will pay off in the long term because, as your system scales, the same tool will simply be called on to do more. You will not only be learning new tools, but will be getting better at applying the tools you have already learned.

Conclusion

Information processing is a huge field with plenty of work for everyone. We as developers can choose to hold defensively to an old model that works for us or vigorously pursue the ideas just described. Either way, we might experience some level of personal and professional success. Plenty of us—and plenty of organizations—will survive using every choice described here…good or bad.

If we fearfully cling to our old ways of doing things, we will certainly not see our jobs getting any easier as time passes. Or they may disappear entirely. Pain is suffered by different professionals in different industries at different times. It's true that: "If nothing changes, nothing changes. If you keep doing what you're doing, you're going to keep getting what you're getting. You want change, make some."[2]

The value we bring to our organizations comes from solving business problems, not from wiring components together. The more our tools and processes allow us to focus on producing that value, the better our careers will be. Agility isn't just for software; it's for us too.

2 Courtney S. Stevens, *The Lies About Truth*

We hope that the material in this report has helped inform your ongoing conversations with your fellow developers, as well as with DBAs, system administrators, and others in your organization.

Let's all listen to what the data is saying!

About the Author

As a support engineer for MarkLogic Corporation, **Eric** helps clients deploy and maintain solutions that manage billions of documents with hundreds of distributed servers that are geographically clustered in multiple countries. The systems he supports serve critical roles in healthcare, finance, and both military and business intelligence communities.

A life-long learner Eric has worked in information technology using Ruby on Rails, integrated circuit design using VHDL and Verilog, embedded controls using C and C++ and business development in retail, consulting and merger and acquisition areas, before embracing multi-model database development. The alignment of this new DBMS with the extraordinary capabilities of commodity hardware was first to capture his interest, while the similarities to the hierarchical and object-oriented databases of past generations then drew him in further.

This report is his sincere effort to communicate that excitement and sense of possibility to others in various technology fields.