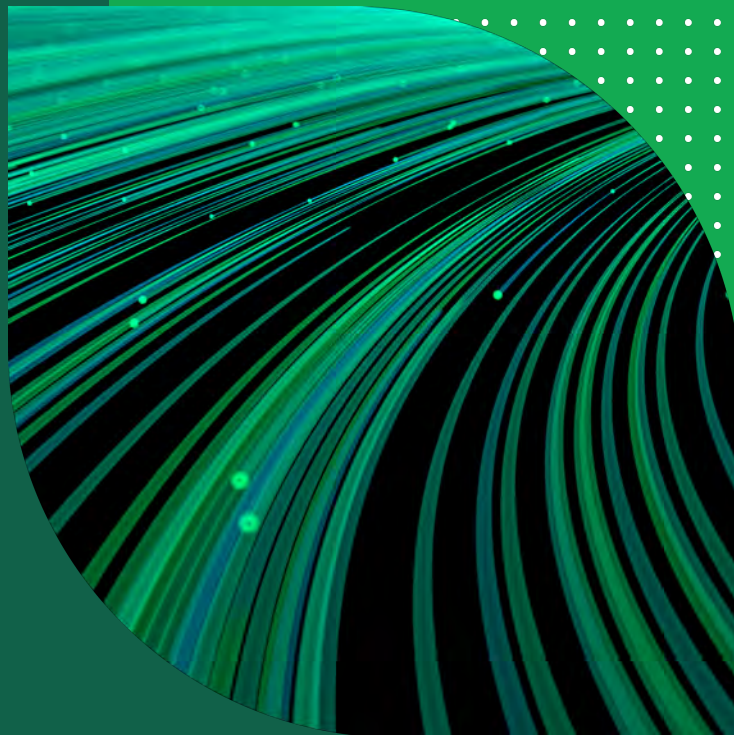
EBOOK
DESIGN

GIANT SPOON

MAY 12, 2020

Developer Autonomy And The Document Model

MAY 8, 2020



mongoDB.

CONTENTS


INTRO
1

The document data model has become the most popular and widely used alternative to the tabular model used by traditional relational databases.

IT'S SO POWERFUL THAT EVEN RELATIONAL DATABASES ARE TRYING TO EMULATE IT.

What initially captures developers' attention is how intuitive documents are. Documents map directly to objects in code, so they are much more natural and efficient to work with, driving higher development velocity. There is no need to decompose data across tables, run expensive JOINs to reassemble objects or integrate a separate ORM layer. Data that is accessed together is stored together, so developers have less code to write and their users get higher performance.

After ease-of-use, what hooks developers on the document model is how flexible it is, giving them faster evolution in the face of constantly changing business requirements.

Beyond the data model, being able to rely on a highly resilient and massively scalable distributed database layer also gives developers a data foundation that they can depend upon.

What all of this adds up to is much greater autonomy for developers to move at speed. This autonomy is critical as more and more organizations adopt microservices patterns to decompose monolithic applications into independently deployable units developed by small, self-sufficient teams.


FLEXIBILITY
=
FREEDOM
+
2
DEVELOPER AUTONOMY
6

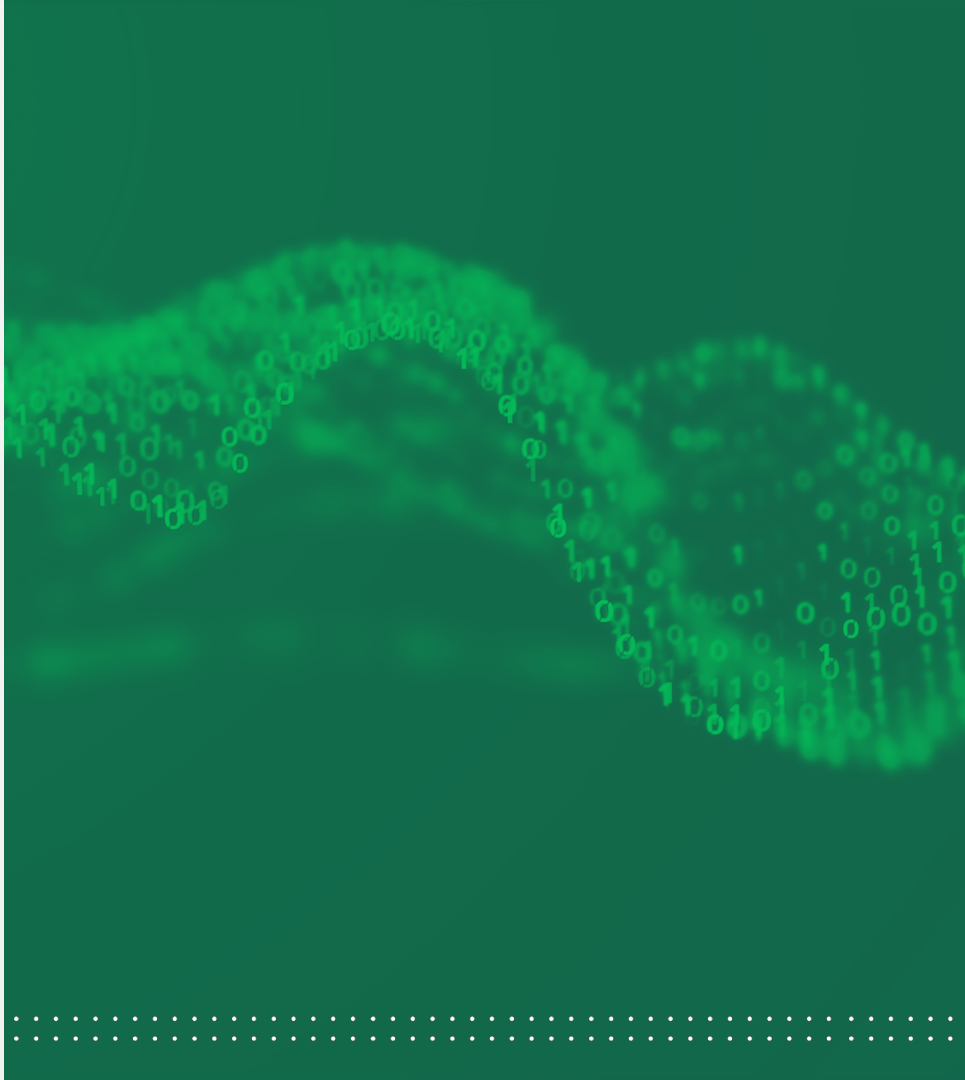
I just heard from a customer who basically mandated that all new development work happen on MongoDB **given how much easier it is** for distributed development teams to develop apps **versus other platforms where it's hard for those teams to work autonomously.**

— **MongoDB Leadership**

Digging deeper, the document model's flexibility eliminates the complex inter-group dependencies that have traditionally slowed developers down.

HOW DOES IT DO THIS?

First, let's take a step back and look at what it's like to work with a traditional relational database.



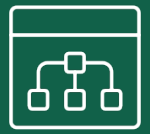
COMING FROM A RELATIONAL WORLD

A relational database's rigid row-and-column structure creates a mismatch between how developers think of – and code – data, and how they need to store it. Developers must coordinate with database administrators (DBAs) to translate the application's rich data schema to fit the rules of the relational model.

Adding an ORM layer partially abstracts the discrepancies between the structure of the data in code versus the data in the database. However, another layer adds complexity for developers and DBAs, requiring more interactions and design

debate. At the end of all of this, ORMs often produce sub-optimal query performance and developers still require deep SQL skills to handle more complex queries.

As the application evolves, so do the data storage and retrieval requirements. Schema modifications, even small ones, require navigating a complex dependency chain, including the need to update ORM class-table



mappings, recompile programming language classes, modify application code, and of course, change and add queries. This necessitates the involvement of multiple teams. Deploying schema migrations can then take the database offline, or at best result in degraded performance while the migration is in-flight.

All of this imposes huge friction to the development process when building applications and adding new features.

HOW MUCH OF A PROBLEM IS THIS… REALLY?

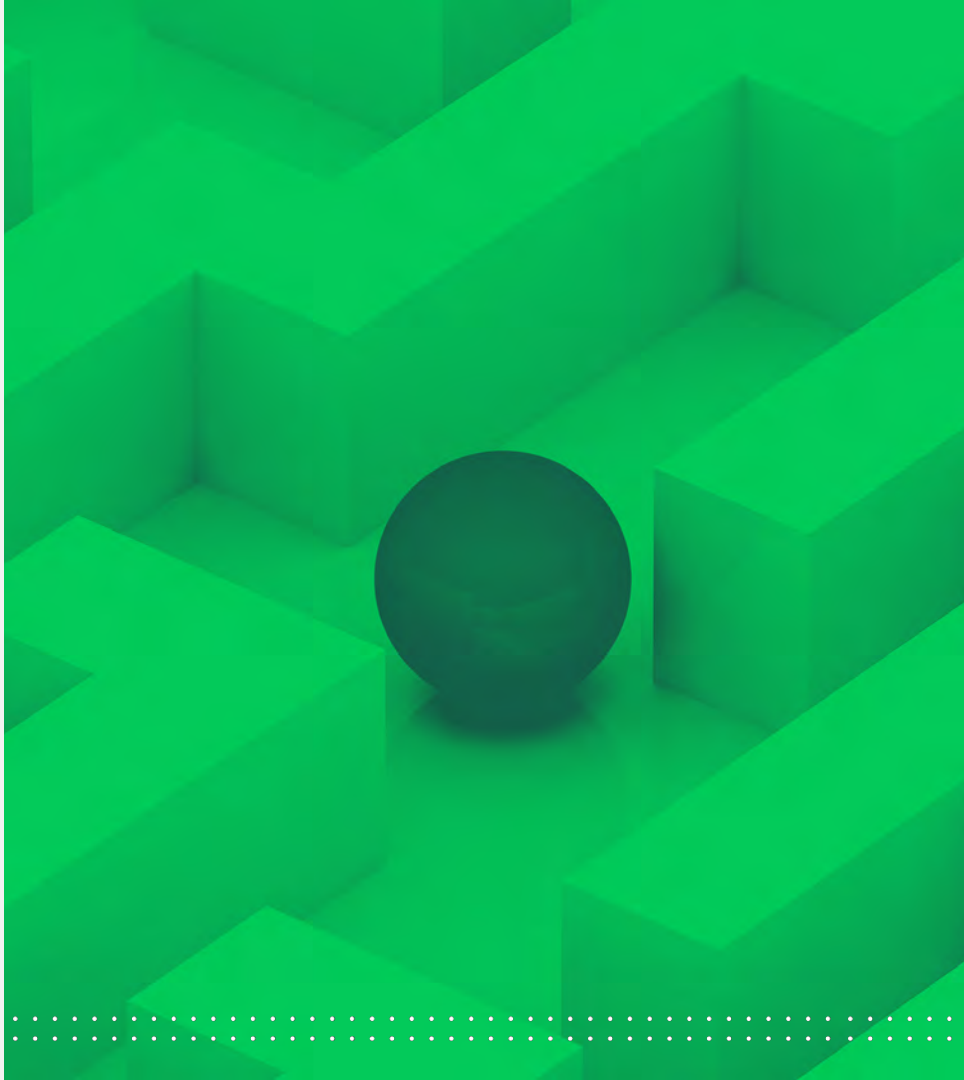
A RECENT SURVEY ESTIMATED

+ That around 60% of all application changes **require modifications** to an existing schema
+ And that **database changes take longer to deploy** than the application changes they are designed to support
+ 43% of respondents reported they are **releasing changes daily or weekly**
+ That they **lose hours or days** reviewing database change scripts
+ Even after these reviews, 84% had **serious production issues** due to database change errors
+ 88% took **more than an hour to resolve** these issues

LINK

Empirically, the continual need to resolve conflicts between the data model in an ever-changing application versus the data model in its relational database increases production problems and adds a lot of extra coordination, slowing the developers and business down.

SO YES, IT'S A **BIG PROBLEM.**



ENTER THE DOCUMENT WORLD

A key attribute of documents is that they are flexible and adaptable. What does this mean and what benefits does this bring?

1. DOCUMENTS ARE SELF-DESCRIBING

There is no need to declare the structure of documents to the database. Developers can start writing code and persist objects as soon as they are created. No convoluted upfront schema design where developers try to map the schema of objects in their application to tables managed by a DBA. And ORMs become a thing of the past.

2. DOCUMENTS ARE DYNAMIC

If a new field needs to be added, it can be created without affecting all other documents in the collection, without updating a central system catalog and without taking the database offline. When you need to make changes to the data model, the document database continues to store the updated objects without the need to perform costly ALTER TABLE operations, update a separate ORM middleware layer, and coordinate all of these changes across multiple developer, DBA, and ops teams. Documents allow multiple versions of the same schema to exist in the same tablespace. Old and new applications can co-exist.

3. DOCUMENTS ARE POLYMORPHIC

Fields can vary from document to document within a single collection (analogous to a table in a relational database). This makes it very easy for developers to model diverse attributes, elegantly handling data of any structure, without having to overload rich structures into rigid rows and columns.

DOCUMENTS ARE FLEXIBLE & ADAPTABLE

WHAT DOES THIS MEAN AND WHAT BENEFITS DOES THIS BRING?

1 DOCUMENTS ARE SELF-DESCRIBING

2 DOCUMENTS ARE DYNAMIC

3 DOCUMENTS ARE POLYMORPHIC

DOCUMENTS ARE **SELF-DESCRIBING**

There is no need to declare the structure of documents to the database. Developers can start writing code and persist objects as soon as they are created. No convoluted upfront schema design where developers try to map the schema of objects in their application to tables managed by a DBA. And ORMs become a thing of the past.

DOCUMENTS ARE **DYNAMIC**

If a new field needs to be added, it can be created without affecting all other documents in the collection, without updating a central system catalog and without taking the database offline. When you need to make changes to the data model, the document database continues to store the updated objects without the need to perform costly ALTER TABLE operations, update a separate ORM middleware layer, and coordinate all of these changes across multiple developer, DBA, and ops teams. Documents allow multiple versions of the same schema to exist in the same tablespace. Old and new applications can co-exist.

DOCUMENTS ARE **POLYMORPHIC**

Fields can vary from document to document within a single collection (analogous to a table in a relational database). This makes it very easy for developers to model diverse attributes, elegantly handling data of any structure, without having to overload rich structures into rigid rows and columns.

While a flexible schema is a powerful feature, there are situations where you might want more control and governance over the data structure and content of your documents.

SCHEMA VALIDATION IN MONGODB GIVES YOU THIS BALANCE.

Through these benefits, the flexibility of the document data model is well suited to the demands of today's agile and DevOps practices. In this world application changes are continuously being deployed into production, driven by small, self-contained autonomous teams.


SHOW ME
THE PROOF
+
3
DEVELOPER AUTONOMY
21

TRAVELERS

Travelers Insurance moved from traditional relational databases to MongoDB as part of a broader microservices and DevOps architecture redesign. To quote Senior Architect Jeff Needham

That was really the first evolution towards MongoDB, which just seems so logical. If we are letting a single team own their database change, why not let the team make the database changes themselves? **But you can't do that with Oracle or SQL Server, you still have to have this DBA skill set to manage schema and manage database changes.** So, for us, looking at the options…

…MongoDB was right in line with **continuous delivery**. I can make database changes as a developer, I don't have to hand that off to another team. It's really about speed. By limiting that work in progress, **we are not waiting on hand offs, we are not waiting on other areas.**

— TRAVELERS

In its session entitled “Bye Bye Legacy: Simplifying the Journey” HSBC developers and architects discussed the benefits of moving to documents and MongoDB as they modernized their trading systems around microservices. It enabled them to shift traditional siloed teams, segregated by management hierarchy, into much more cross functional groups with self-sufficient, multi-disciplinary pods.

TOYOTA

MATERIAL HANDLING

Principal System Architect and IT-Manager Filip Dadgar at Toyota Material Handling Europe discussed why the company had selected MongoDB running on the fully-managed Atlas cloud service for its new smart-factory IoT project constructed around a microservices architecture.

The most beautiful part is the data model. Everything is a natural JSON document. So **for the developers, it is easy, really easy for them to work with quickly**. Spending time on building business value, rather than data modeling.

— TOYOTA
MATERIAL HANDLING

GETTING STARTED

4

Take a look at our What is a Document Database page to learn more about what documents can do for developer productivity and how they enable the autonomy needed for efficiency and velocity.

You can also try out the power of documents for free with our sample data in MongoDB Atlas.

TRY IT NOW

mongoDB®

THANK YOU.

GIANT SPOON