mongoDB®

Productivity, Autonomy, and the Document Model

Productivity, Autonomy,
and the Document Model

Frustrated with your pace of innovation?

It might be your database.

Speed, agility and efficiency of application development has never been more critical than right now. Companies of all sizes, in all industries, are racing to react to sudden shifts in customer demands and competitive dynamics. But many are building or maintaining applications on top of aging, inflexible relational data platforms that cripple their productivity.

There is a better way. 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 easy and intuitive it is to work with documents. Because they map directly to objects in code, documents offer a more natural approach to development, and drive faster development cycles as a result. Data that is accessed together is stored together, so developers have less code to write and end-users get better performance.

After ease-of-use, what hooks developers on the document model is flexibility and reliability. In a world of constantly changing business requirements, the document model facilitates rapid iteration. And because the data model is built upon a highly resilient and massively scalable distributed database layer, it gives developers a data foundation that they can depend upon, no matter how their needs change.

The document model allows developers to move with speed and autonomy, which is critical as more organizations adopt microservices patterns to decompose monolithic applications into independently deployable units developed by small, self-sufficient teams.

Coming from a Relational World

A relational database's rigid row-and-column structure does not match the way developers think about, or work with, data. And it slows them down.

When working with a relational database, developers must constantly coordinate with database administrators (DBAs) to translate the rich data schemas of modern applications to fit the rules of the relational model.

Requiring object-relational mapping (ORM) adds a layer of abstraction between the structure of the data in code versus the data in the database. And it adds complexity for developers and DBAs, requiring more interactions and introducing the potential for design debate. At the end of all of this, ORMs often produce sub-optimal query performance, and developers still need deep SQL skills to handle more complex queries.

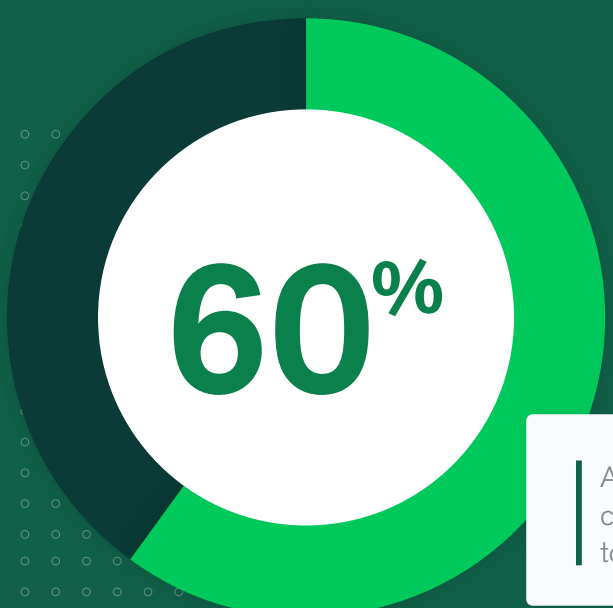
As an application evolves, so do the data storage and retrieval requirements. Schema modifications, even small ones, require navigating a complex dependency chain, including the need to update ORM class-table mappings, recompile programming language classes, modify application code, and of course, change and add queries. This necessitates the involvement of multiple teams. And deploying schema migrations can sometimes take the database offline, or result in degraded performance while the migration is in-flight.

All of which adds friction to the development process when building applications and adding new features.

> "After we switched to remote work during the pandemic, we pretty much mandated that all new development work happen on MongoDB. It's much easier for distributed development teams to develop apps versus other platforms that make it hard for teams to work autonomously."
>
> CIO, Fortune 500 company

Is this really a big problem?

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.



Around 60% of all application changes require modifications to an existing schema.

Consider also that in the same survey:

- **43%** of respondents reported they are releasing changes daily or weekly.
- Many respondents claim to lose **hours** or **days** reviewing database change scripts.
- **84%** had serious production issues due to database change errors, even after reviews.
- **88%** took more than an hour to resolve these issues.

Empirically, the continual need to resolve conflicts between a relational data model and an ever-changing application increases production problems and adds complication, slowing down developers, and the businesses they support.

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 are the benefits?

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. There is 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 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 dynamic

If a new field needs to be added, it can be created without affecting all other documents in the collection, and without needing to update a central system catalog or 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.

In other words, the flexibility of the document data model is well suited to the demands of today's agile DevOps practices, a world in which application changes are continuously being deployed into production, driven by small, self-contained autonomous teams that move quickly and efficiently.

Command and control

While a flexible schema is a powerful feature, there may come a time in an application's lifecycle – for example when it's functionality has reached steady state – that you want more centralized control over the data structure and content of your database.

With schema validation, you can apply data governance standards to a document schema when an application is in production, without sacrificing the benefits of a flexible data model in development.


mongoDB.

Show me the Proof

TRAVELERS

TOYOTA
MATERIAL HANDLING

Travelers Insurance

Travelers Insurance moved from traditional relational databases to MongoDB as part of a broader microservices and DevOps architecture redesign.

"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."

– Jeff Needham, Senior Architect, Travelers Insurance

HSBC

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.

– View Session Link

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."

– Filip Dadgar, Principal System Architect & IT-Manager, Toyota Material Handling

Getting Started

Ready to increase productivity
and give MongoDB Atlas a try?

Sign up today for free.

Not yet ready to try it out? Have questions? Contact us