Dev.Pro
Dev.Pro Website

5 MongoDB Best Practices for Database Architecture

When you're planning to create a platform that operates with a large amount of configured data or has dozens of business entities, it's important to choose the correct database design at the outset.

Your cloud and DevOps specialists will need to create a dynamic, high-availability database that is flexible enough to accomodate frequent schema change. The design of your future database should be horizontally scalable for better read capacity and should support entity references.

That's where MongoDB, a document-oriented NoSQL database, comes in handy. In this article, we're going to explain what MongoDB is good for and run through some of the leading MongoDB best practices.

Reasons to Use MongoDB Database Design

You might be wondering what MongoDB is used for, or if it will fit your project. MongoDB can be used for both agility and digital transformation, as well as legacy system modernization processes.

You can therefore use MongoDB for IoT products, mobile apps, enterprise systems, content management, and even financial solutions.

Pros of MongoDB Database Design:

- **Document-oriented** — Document-oriented (schema-less) database with a flexible schema design.
- **Replication** — Provides high availability with replica sets.
- **Scaling** — Scales horizontally using sharding.
- **Document versioning** — Powerful aggregation pipeline framework can be used for document versioning.
- **Indexing** — Any fields in a document can be indexed.

Cons of MongoDB Database Design

- **No JOINs** – The whole system will be designed with "joins" on the application layer.

This adds more complexity to the application layer, but reduces the database load. However, MongoDB has lookups that function similarly to JOINs but also affect performance and come with limitations. For example, when you concretize a pipeline for a joined collection, you're not allowed to include either stage in the pipeline field.

- **No foreign keys** – The system will be designed with document relations on the application layer. This adds additional complexity to the application layer and increases the chances of a mistake. It's therefore important to initiate thorough testing that covers all potential cases. While there are several potential downsides, this method also removes the load from the database to control relations during inserts and deletes.

Top 5 MongoDB Use Cases for Database Architecture

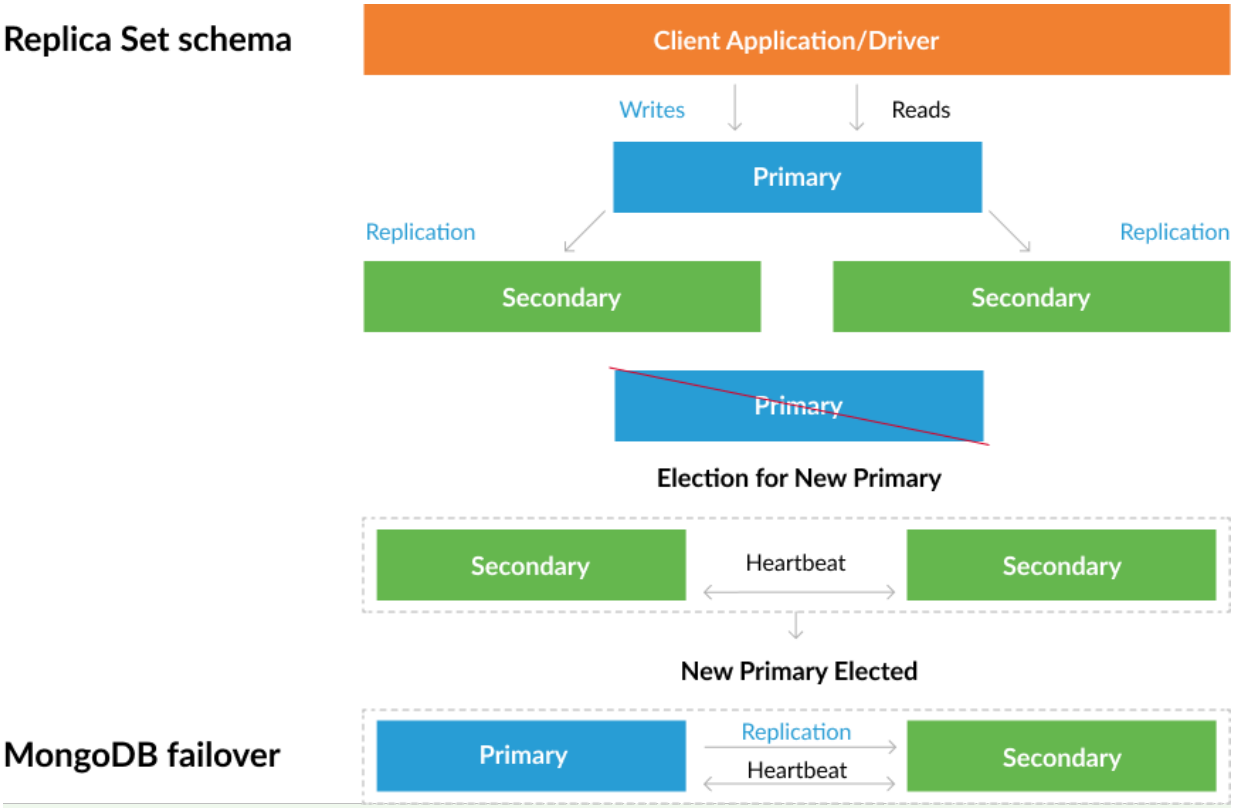
Let's analyze five use cases highlighting how MongoDB can be used in real-world situations.

Use Case #1: MongoDB Database Design Supports Backward Compatibility

Imagine a restaurant with 30 menu items. If you want to add or replace an item, you'd need to reprint the menu entirely. MongoDB's database design allows you to do this with no downtime. This is known as backward compatibility support.

This allows you to create a new field or change an existing field while continuing to support the existing field. This allows a database to work without an upgrade to a new schema design.

Use Case #2: MongoDB Database Design Assures High Availability with Replica Sets



Example of possible replica set configuration

Documentary-heavy businesses like restaurant chains or logistics centers have thousands of documents, processes and people. As a result, they require a high level of protection against data loss and avoidance of maintenance downtime. Moreover, permanent availability of data is also important, since staff must immediately react if, for instance, some products run out. MongoDB helps handle these issues.

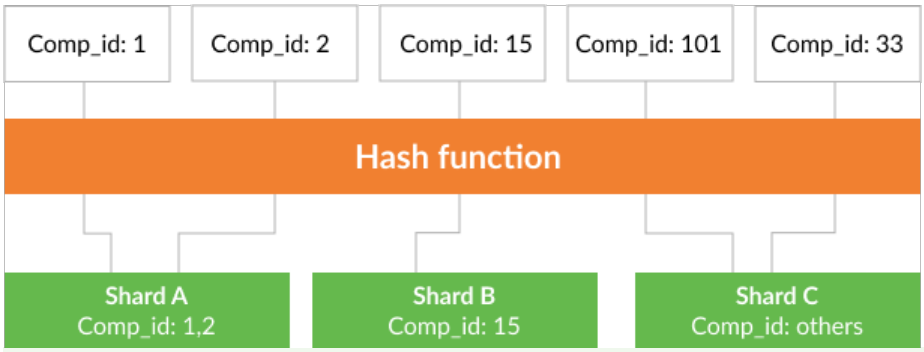
Instead of using the standard “primary-secondary” replication, businesses can use MongoDB’s “replica sets,” which contain multiple instances. These instances consolidate during the work process and offer a high level of access to infrastructure.

How do replica sets work? In the example above, there is a structure that serves as the primary instance, while the other structures are replicas. If the primary instance becomes inaccessible, one of the replicas will automatically replace it, since it’s a built-in process in MongoDB.

Enabling the replica process is quite simple and can be run after adding multiple commands. After installation, replicas will sync automatically. The only situation where you’ll need to spend additional time on this task is when your system operates with a large amount of data.

Use Case #3: MongoDB Database Design Provides Sharding for Horizontal Scaling

MongoDB sharding helps scale horizontally by splitting data and loads between several servers. The load can be distributed through several servers and/or data duplicated to keep the system operating if hardware fails. Selecting the right shard key is essential for equally distributed data.



Example of shard key configuration

What is the role of this feature for database owners? When you launched your application, you probably didn't predict the future load. But as your business grew in popularity, your load needs increased. For example, your online store grew from one small shop to a nationwide chain. Naturally, to prevent system failures, you need to add additional servers. Horizontal scaling using sharding helps you do it on the fly with no downtime.

Use Case #4: MongoDB Database Helps Set Up Reference Management

Let's say your project has 100-plus entities with relations between each other and all relations are configured on the application layer using the same pattern. It can be applied to a restaurant menu where you have price collection and product collection, and during the order process, data can be collected from different sources. Like, tomatoes can be bought from different suppliers at different prices.

If a single-document approach does not match your project needs, you may need to divide your data between multiple documents (e.g. in various collections or databases).

MongoDB applications has two methods for reference management:

- **Manual references** – Use the _id field in the documents as reference to each other.
- **DBRefs** – Use the value of the first document _id field (or collection and database name) for reference to another document.

Typically, a manual reference approach is most suitable if you need to unite a couple of related documents. But if your task is to combine documents from multiple collections, use DBRefs.

Use Case #5: MongoDB Database Design Allows for Document Versioning Using Aggregation Pipeline

Document versioning makes the history of versions inside your system accessible at any time. Instead of using multiple systems to control documents, you can store them all in one database. Document versioning is a helpful solution for highly-controlled business areas, like financial or healthcare, which require a specific point

in time of a dataset.

To facilitate the processing of a large volume of data, MongoDB provides users with an Aggregation Framework. With the help of this tool, you can improve the performance of system queries, since the Aggregation Pipeline supports various embedded features and indexing types.

Let’s review how this works in a real-world example. Let’s say a restaurant uses MongoDB database design. If the unit price for any item changes from week to week, MongoDB lets the restaurant save those changes and delivers the correct unit price on the requested date.

Example of document versioning in action

Summary

MongoDB database design has a variety of strengths, including no-downtime schema changes, high database availability, and easy reference management. Like any solution, MongoDB also has drawbacks.

That’s why, before choosing a solution, we recommend turning to database experts who can share their advice on different options and mitigate risks during development and migration. If you’re looking for advice, turn to Dev.Pro’s cloud experts. We can share our expertise to help you adopt the best practice for your case.