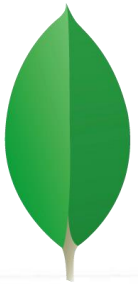
# Introduction to MongoDB



# Challenges with Relational Databases

Adding features to relational databases becomes challenging due to complexity.

Users face difficulties understanding and navigating applications with multiple tables.

As features grow, maintaining efficiency in a bloated, inefficient table is problematic.

The relational approach poses challenges for developers and application users.

Pulling data from multiple places in a relational database results in inefficiency, impacting both speed and user experience.

# Relational Database

Traditional relational databases resemble Microsoft Excel on an enhanced scale.

Rows and columns organize data into tables, forming a rigid structure.

This Excel-like structure creates challenges for evolving data requirements.

# Introduction to MongoDB

MongoDB's document model stores data in flexible BSON (Binary JSON) documents.

This shift is fundamental, diverging from traditional relational databases.

The document-oriented structure is a key design choice, defining MongoDB.

Developers appreciate MongoDB's approach, making it a popular choice.

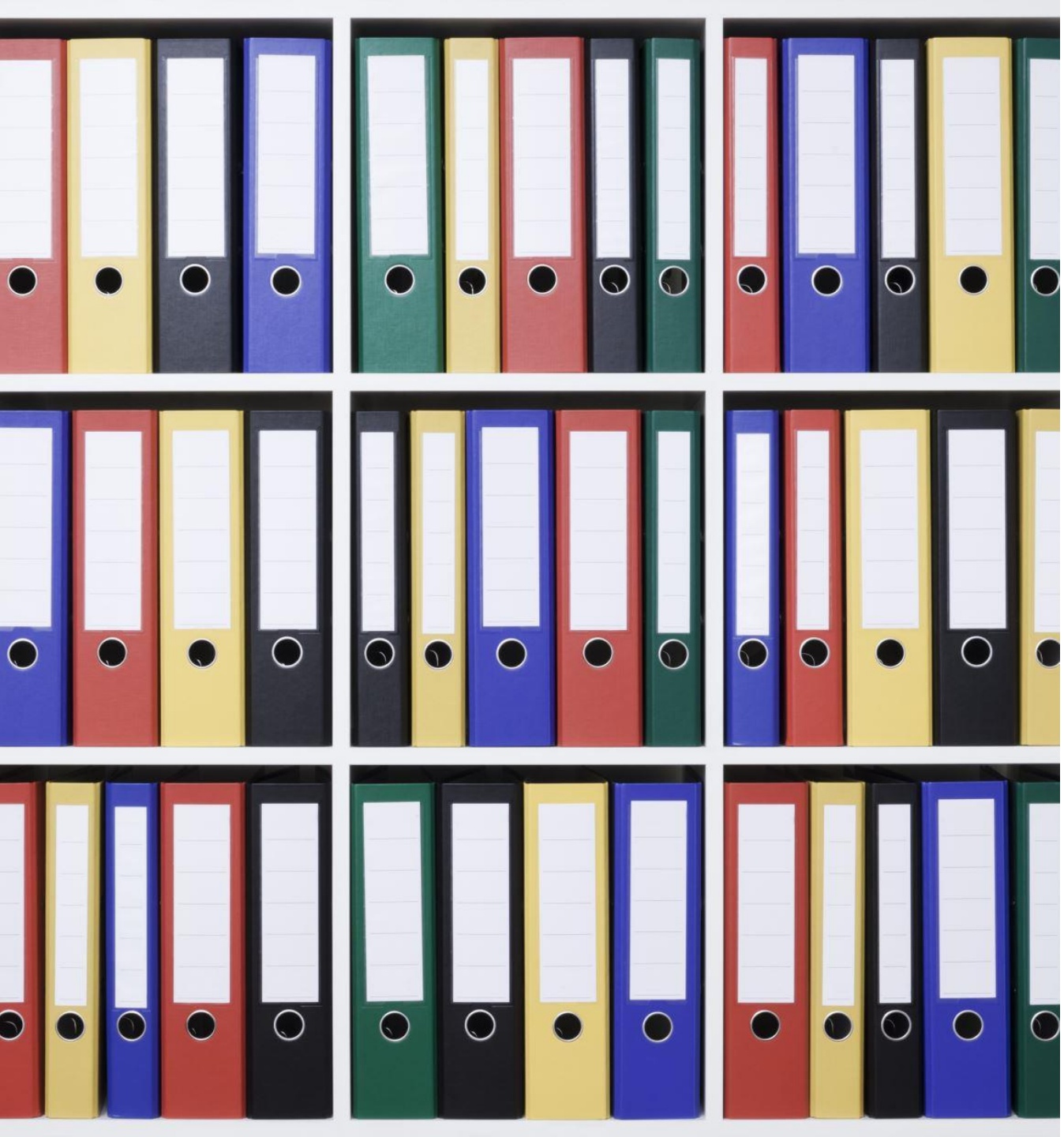
Let's explore how this model differs in practice.

# Introduction to MongoDB

- MongoDB is a NoSQL document database.
- Data is organized as documents, further grouped into collections in MongoDB.
- Scalable High-Performance Open-source, Document-orientated database.
- Built for Speed
- MongoDB is written in C++, JavaScript, and C language

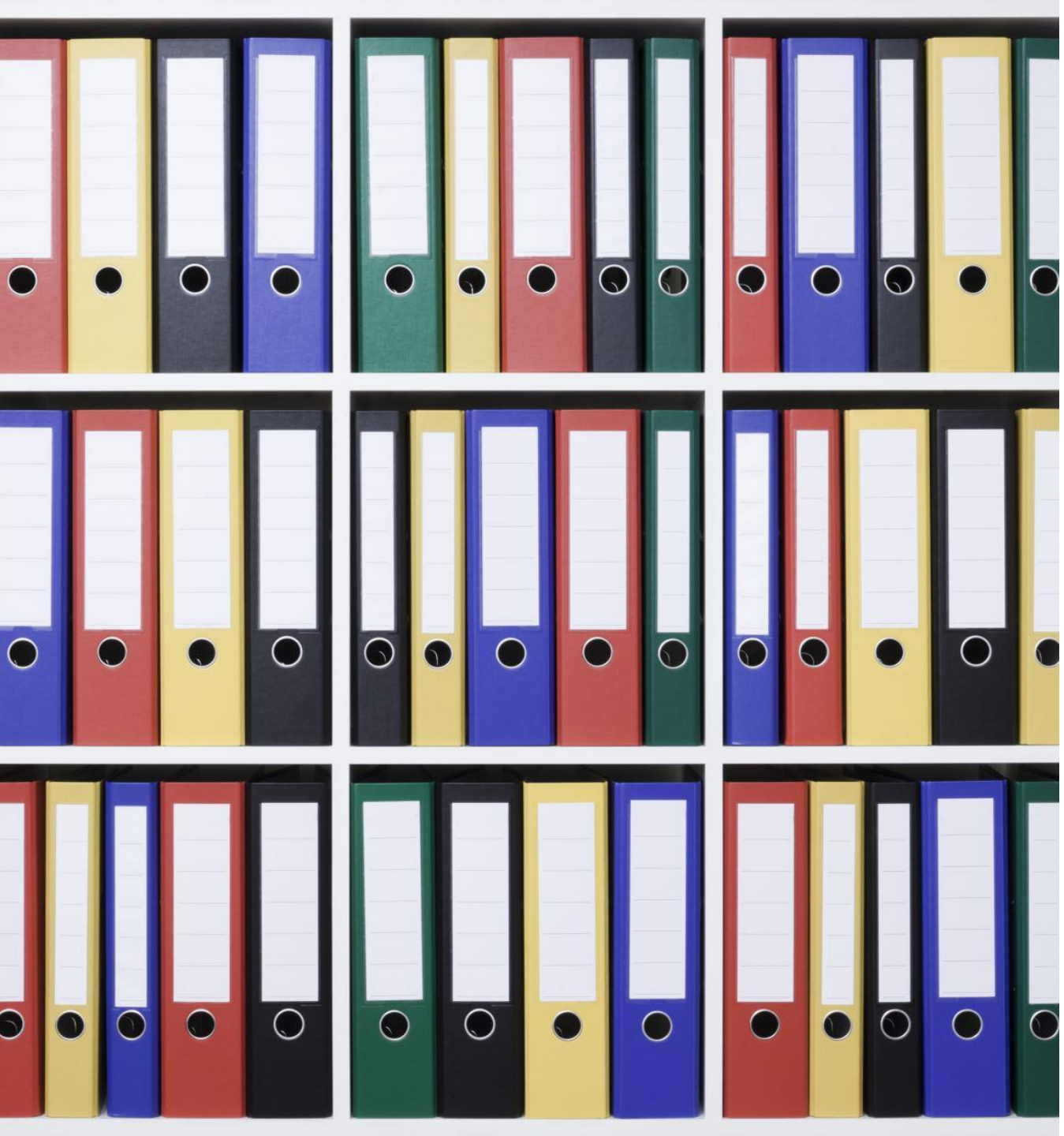
# Introduction to MongoDB

- Rich Document based queries for Easy readability.
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map / Reduce for Aggregation.



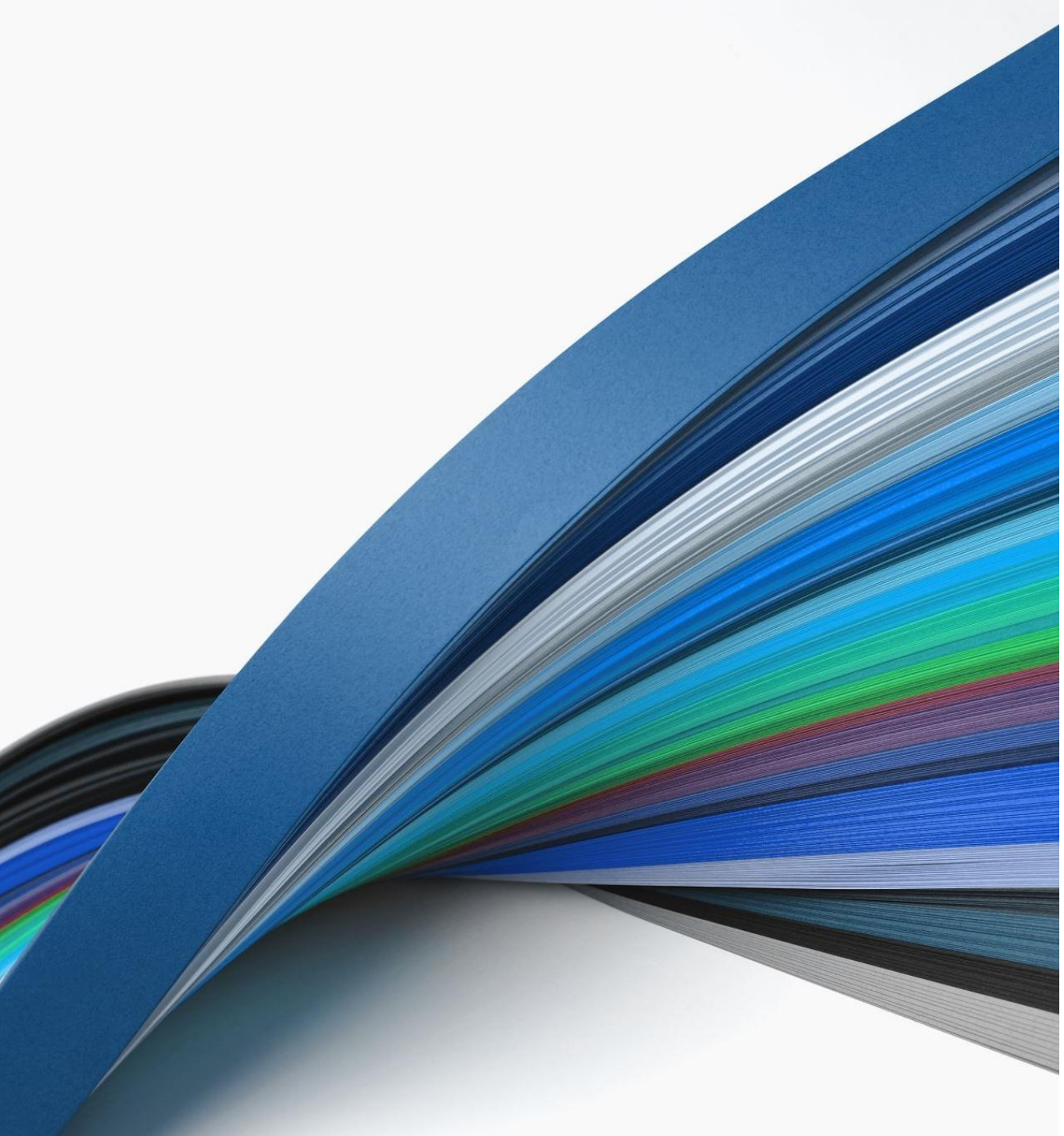
# Document Model Overview

- MongoDB's document-oriented structure is the cornerstone of its design.
- Data is stored in BSON documents, offering flexibility for dynamic changes.
- BSON allows efficient handling of diverse data types in a natural format.
- The document model aligns with JSON-like structures, enhancing readability.
- MongoDB's approach empowers developers with a user-friendly data model.



# Advantages of Document Storage

- MongoDB's document storage mimics the structure of physical medical records.
- Data is organized in a way that is both efficient for computers and readable for humans.
- The document model allows for the inclusion of varying data attributes seamlessly.
- Developers appreciate the power of adapting data structures as needed.
- MongoDB's approach facilitates efficient storage and retrieval of diverse data.



# Flexibility for Developers

- MongoDB empowers developers by accommodating the needs of their applications.
- Developers can store data in a natural way without rigid schema constraints.
- This flexibility ensures that MongoDB adapts to the evolving needs of applications.
- MongoDB's document model simplifies the development process.
- Adding or modifying data becomes a straightforward task for developers.

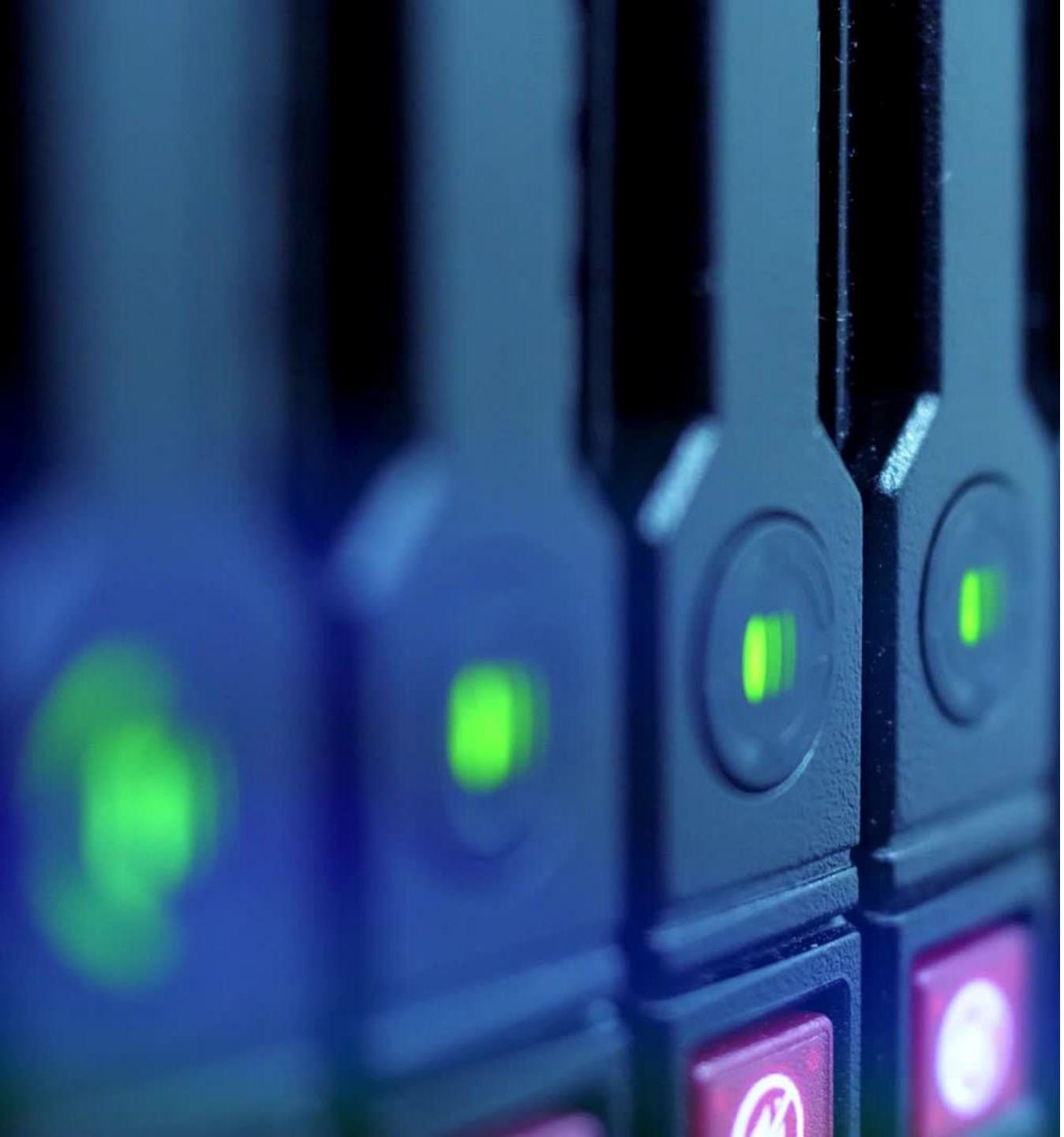
# MongoDB's Distributed System

Coordination of Multiple Servers

- MongoDB excels in fault tolerance by keeping redundant copies across servers.
- A single server failure does not disrupt the overall application.
- The distributed nature of MongoDB allows for horizontal scaling with ease.
- Data coordination across multiple servers ensures high availability.
- MongoDB's distributed system architecture is a key advantage over traditional databases.

# Fault Tolerance in MongoDB

- MongoDB's native fault tolerance is achieved through redundant data copies.
- In case of a server failure, the system automatically switches to a functioning server.
- This fault tolerance feature enhances the reliability of MongoDB in real-world scenarios.
- Applications remain resilient, even in the face of unexpected server issues.
- MongoDB's fault tolerance is a critical aspect for robust and uninterrupted operations.



# Horizontal Scaling with Sharding

Sharding for Scalability

- MongoDB's sharding capability facilitates horizontal scaling for data growth.
- Adding more servers is a seamless solution for scaling MongoDB horizontally.
- Sharding ensures that data is distributed effectively across different instances.
- MongoDB's approach to scalability avoids the need for costly hardware upgrades.
- Horizontal scaling is particularly advantageous for cloud environments.

Horizontal scaling involves adding more machines or nodes to a system, while vertical scaling involves adding more power (CPU, RAM, storage, etc.) to an existing machine.

# Horizontal scaling Vs Vertical scaling

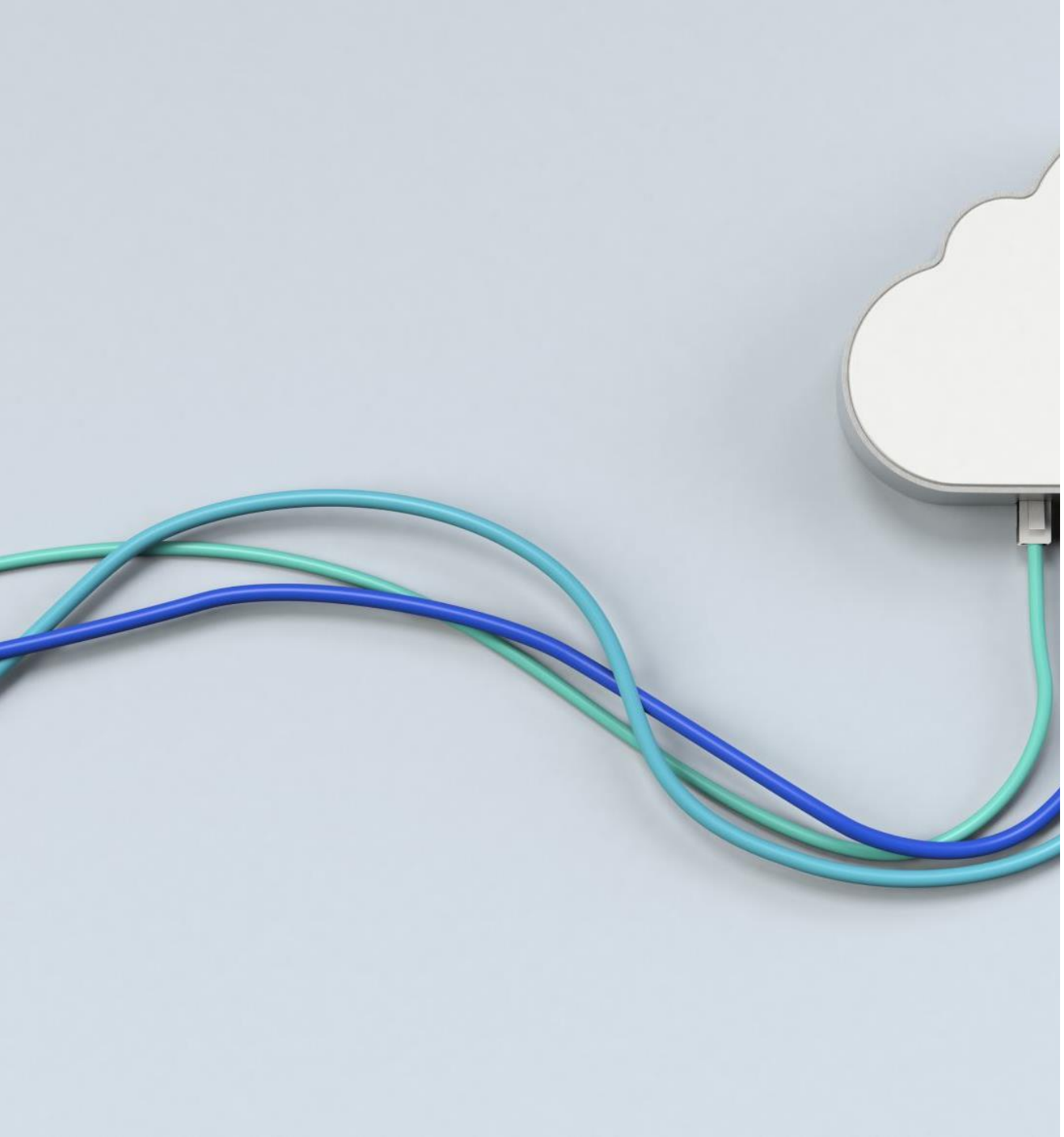
# MongoDB for Real-time Access

Efficient Data Access

- MongoDB allows data to be moved globally, ensuring fast access for users.
- Real-time access to data is achieved through MongoDB's distributed architecture.
- Data can be strategically placed near users, optimizing access times.
- MongoDB's architecture is well-suited for applications requiring real-time responses.
- Efficient data access is a key strength in MongoDB's design.

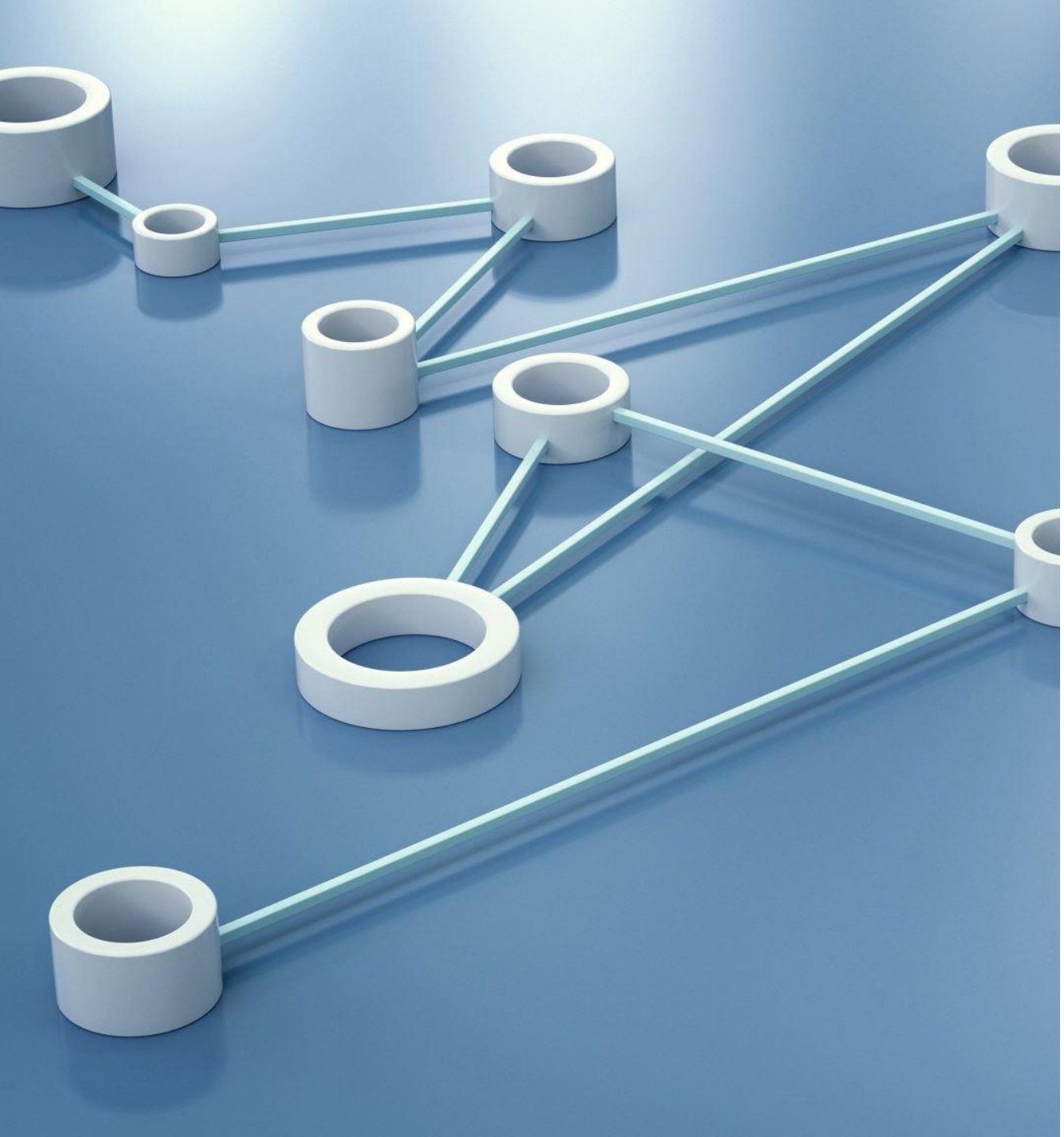
# Management Tools for MongoDB

- MongoDB offers a suite of management tools for various aspects of database operations.
- Provisioning tools help set up and configure MongoDB clusters effectively.
- Security features, monitoring, and backup tools are critical for robust database management.
- MongoDB's management tools are tailored to meet the demands of high-performance use cases.
- These tools ensure operational excellence and adherence to strict SLAs.



# MongoDB for the Cloud

- MongoDB is designed to excel in cloud environments, aligning with modern computing trends.
- MongoDB Atlas, the database-as-a-service offering, provides worry-free management.
- Availability on AWS, Azure, and Google Cloud Platform offers flexibility to users.
- MongoDB's cloud-centric design is in line with the preferences of many organizations.
- The move toward cloud-based services is driving MongoDB's widespread adoption.



# MongoDB Atlas

- MongoDB Atlas is a fully elastic database-as-a-service solution.
- It simplifies database management by handling configurations, security, and monitoring.
- Availability on major cloud platforms gives users the freedom to choose providers.
- MongoDB Atlas ensures users can focus on application development without database management worries.
- The platform aligns with the need for scalable and reliable cloud-based databases.

# MongoDB in Real-world Applications

- MongoDB finds applications in renowned companies such as Toyota, SAP, and Adobe.
- Widely deployed in areas like IoT, mobile applications, and real-time analysis.
- MongoDB's versatility makes it a preferred choice for handling large-scale data requirements.
- The adoption by major corporations highlights MongoDB's reliability and effectiveness.
- Real-world success stories emphasize MongoDB's impact on various industries.

# MongoDB Security Features

- Authentication and authorization features.
- Native data encryption for enhanced protection.

# MONGODB

**Document**

A way to organize and store data as a set of field-value pairs

```
{
  <field> : <value>,
  <field> : <value>,
  "name"  : "Lakshmi",
  "title" : "Team Lead",
  "age"   : 26
}
```

# What is MongoDB great for?

RDBMS replacement for Web Applications.

Semi-structured Content Management.

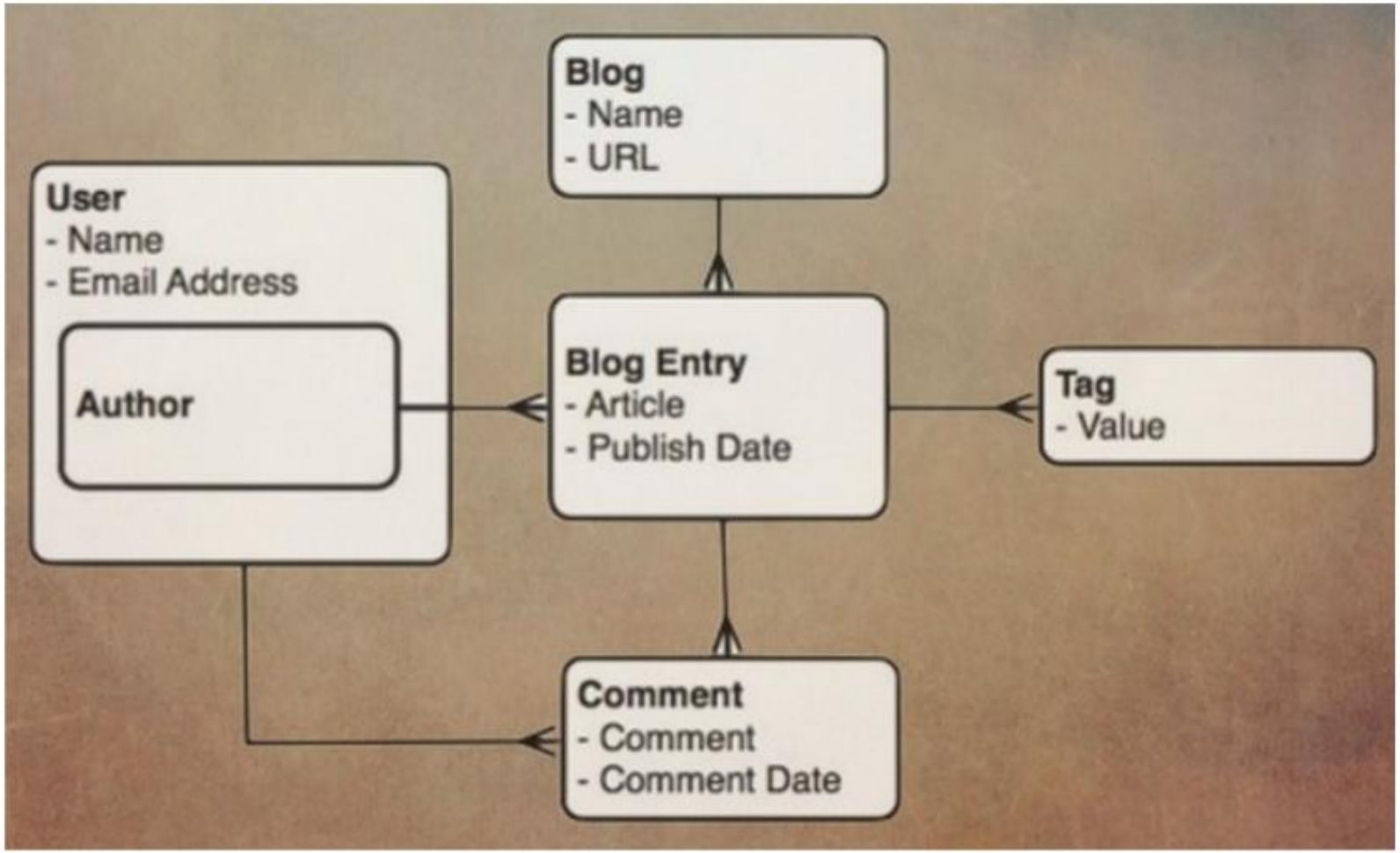
Real-time Analytics & High-Speed Logging.

Caching and High Scalability

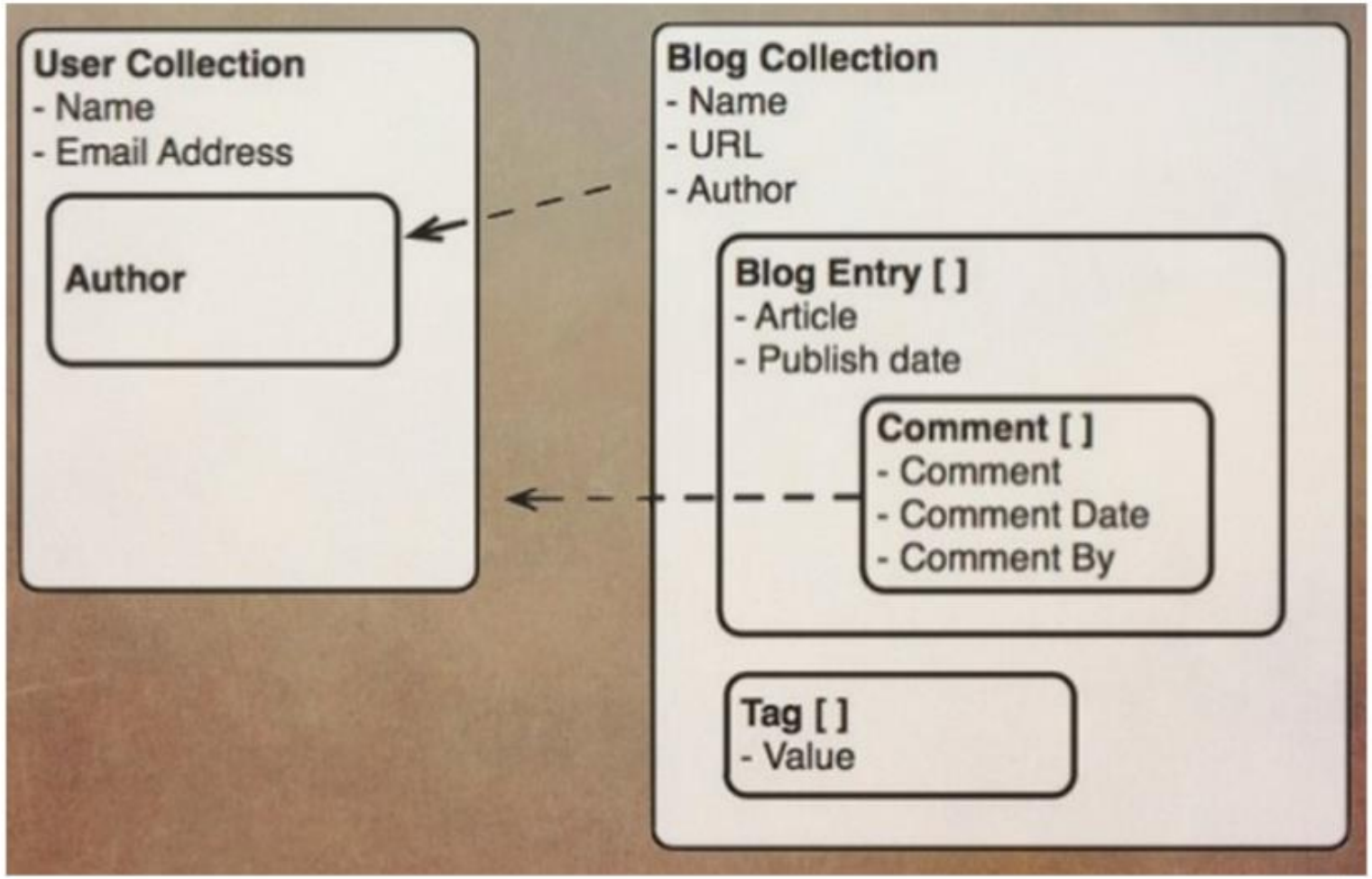
Web 2.0, Media, SAAS, Gaming HealthCare, Finance, Telecom, Government

# Let’s Dive in !

# Blog in relational DB



# Blog post structure in document DB



# Blog post in JSON DB

```
{  _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
   author : "steve",
   date : "Sat Apr 24 2013 19:47:11",
   text : "About MongoDB...",
   tags : [ "tech", "databases" ],
   comments : [
          {
              author : "Fred",
              date : "Sat Apr 25 2013 20:51:03 GMT-0700",
              text : "Best Post Ever!"
          }
    ]
}
```

**When I say**

# Database

**Think**

# Database

- Made up of Multiple Collections.
- Created on-the-fly when referenced for the first time.

## When I say

# Collection

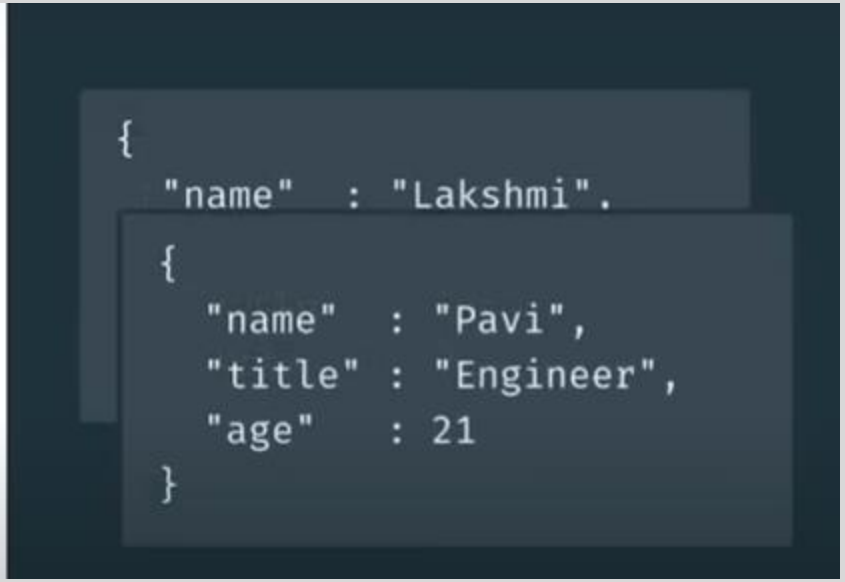
## Think

# Table

- Schema-less, and contains Documents.
- Indexable by one/more keys.
- Created on-the-fly when referenced for the first time.
- Capped Collections: Fixed size, older records get dropped after reaching the limit.

## Collection

An organized store of documents in MongoDB, usually with common fields between documents



| Query | Description |
| --- | --- |
| db.collection.find({}) | Retrieve all documents in the collection. |
| db.collection.find({ field: value }) | Retrieve documents matching specified field-value pair. |
| db.collection.insertOne({}) | Insert a single document into the collection. |
| db.collection.insertMany([{}, {}]) | Insert multiple documents into the collection. |
| db.collection.updateOne({ filter }, { $set: { field: value } }) | Update a single document based on a filter. |
| db.collection.updateMany({ filter }, { $set: { field: value } }) | Update multiple documents based on a filter. |
| db.collection.deleteOne({ filter }) | Delete a single document based on a filter. |
| db.collection.deleteMany({ filter }) | Delete multiple documents based on a filter. |
| db.collection.aggregate([pipeline]) | Perform aggregation operations on the collection. |

```
var p = {
'_id': '3432',
'author': DBRef('User', 2),
'title': 'Introduction to MongoDB',
'body': 'MongoDB is an open sources.. ',
'timestamp': Date('01-04-12'),
'tags': ['MongoDB', 'NoSQL'],
'comments': [{'author': DBRef('User', 4),
              'date': Date('02-04-12'),
              'text': 'Did you see.. ',
              'upvotes': 7, ... ]
}
> db.posts.save(p);
```

# Understanding the Document Model.

Create Index on any field in the document

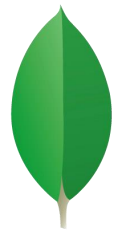
```
// 1 means ascending, -1 means descending
> db.posts.ensureIndex({'author': 1});

//Index Nested Documents
> db.posts.ensureIndex('comments.author': 1);

// Index on tags
> db.posts.ensureIndex({'tags': 1});

// Geo-spatial Index
> db.posts.ensureIndex({'author.location': '2d'});
```

# Secondary Indexes

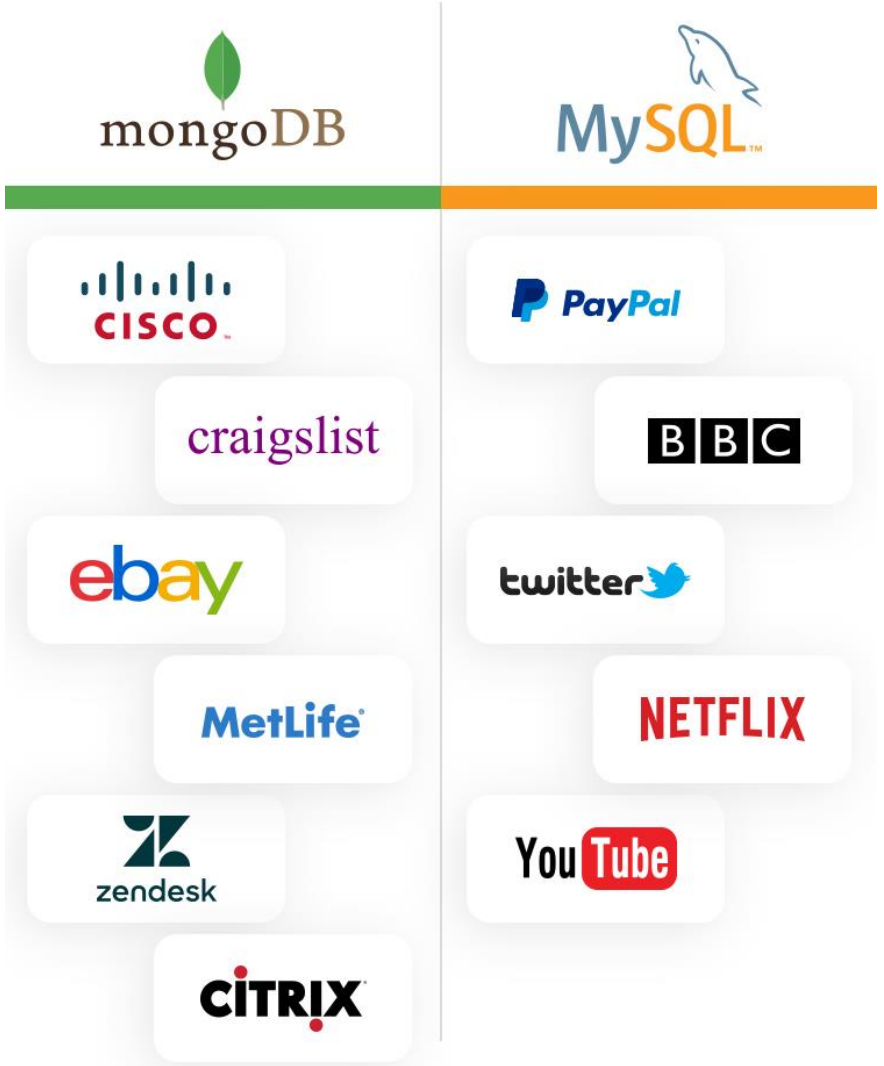
# What about Queries? So Simple

```
// find posts which has 'MongoDB' tag.
> db.posts.find({tags: 'MongoDB'});

// find posts by author's comments.
> db.posts.find({'comments.author':
DBRef('User',2)}).count();

// find posts written after 31st March.
> db.posts.find({'timestamp': {'gte': Date('31-03-12')}});

// find posts written by authors around [22, 42]
> db.posts.find({'author.location': {'near':[22, 42]}});
```

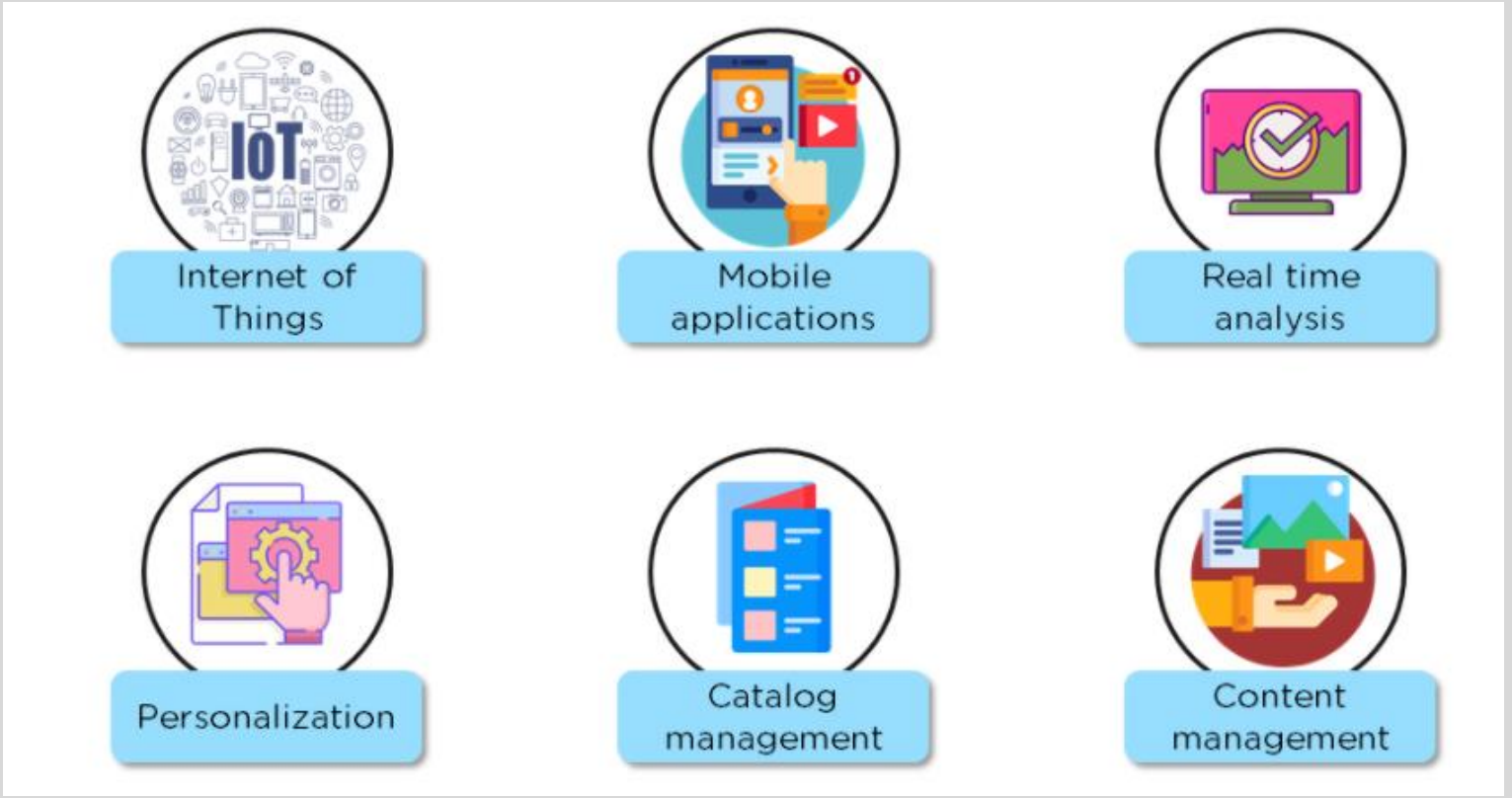
$gt, $lt, $gte, $lte, $ne, $all, $in, $nin, count, limit, skip, group, etc…



| mongoDB | MySQL |
| --- | --- |
| **Use Case** | **Use Case** |
| Real-time analytics, content management, IOT, mobile apps | Legacy applications or applications that require multi-row transactions |
| **Data Structure** | **Data Structure** |
| No schema definition required | Structured data with clear schema |
| **Risk** | **Risk** |
| Less risk of attack due t design | Risk of SQL injection attacks |
| **Analysis** | **Analysis** |
| MongoDB works great for unstructured data and lends you opportunity for growth. | MySQL is a perfect when your data is structured and you are in need of a traditional relationship database. |

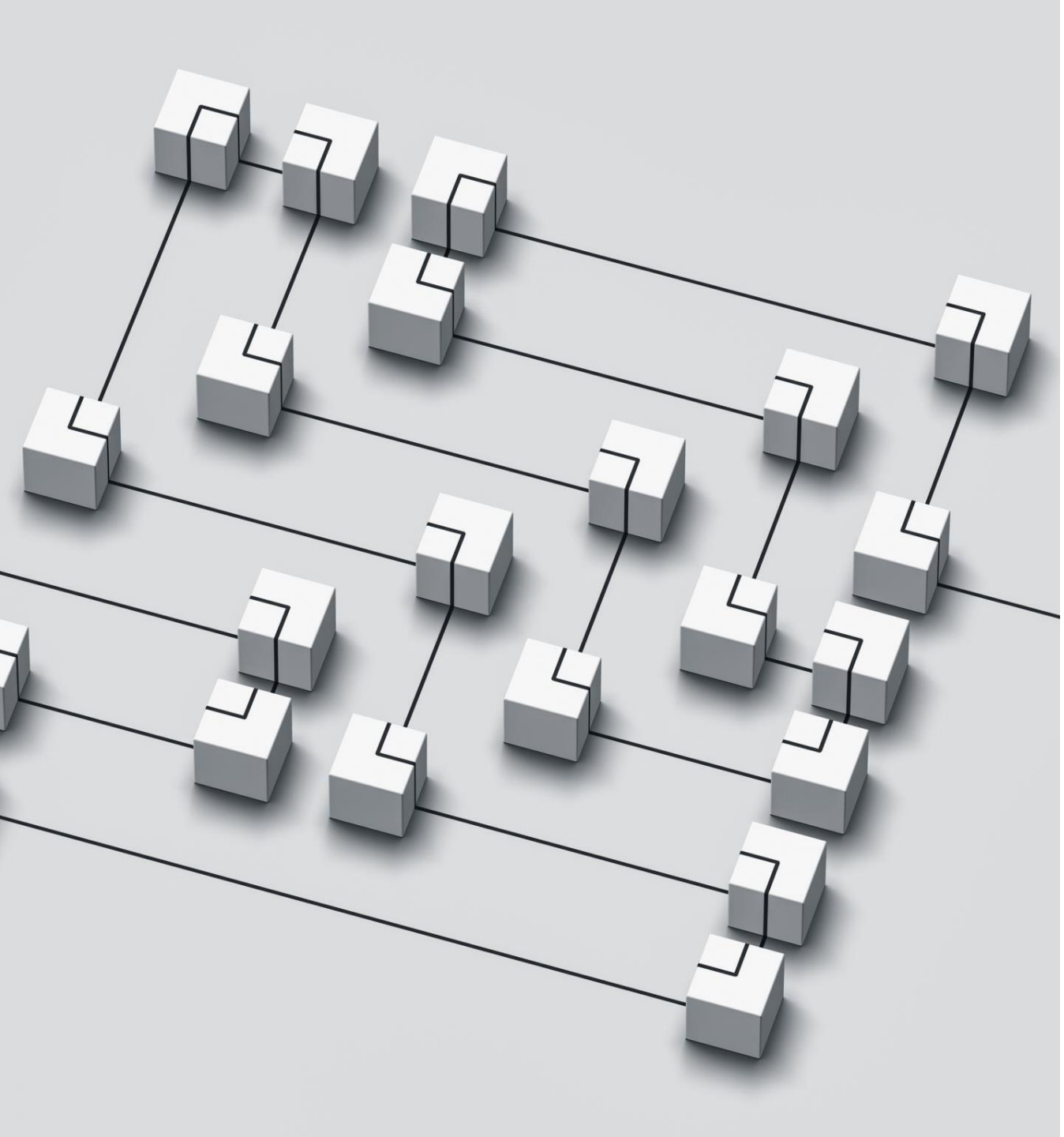

mongoDB
MySQL
cisco
PayPal
craigslist
BBC
ebay
twitter
MetLife
NETFLIX
zendesk
YouTube
CITRIX

# MongoDB Applications



| Feature | Relational Database | MongoDB |
| --- | --- | --- |
| Data Structure | Tables with rows and columns | Collections with flexible BSON documents |
| Schema | Fixed schema | Dynamic schema, adaptability over time |
| Query Language | SQL | MongoDB Query Language (MQL) |
| Flexibility | Rigid structure | Flexible, accommodates evolving data |
| Relationships | Complex relationships | Simple and nested structures |
| Scalability | Vertical scaling | Horizontal scaling with sharding |
| Complexity | High | Lower, especially for developers |
| Use Cases | Traditional applications | Diverse applications, big data, IoT |
| Key Strengths | ACID transactions | High performance, scalability, flexibility |
| Data Retrieval Efficiency | Joins required | Efficient retrieval, no need for joins |
| Schema Evolution | Challenging | Seamless, supports dynamic changes |
| Examples | MySQL, PostgreSQL | MongoDB |

# COMPARISON - RELATIONAL DATABASE VS. MONGODB



# MongoDB's Advantages Over Relational Databases

- MongoDB's combination of the document model and distributed systems offers a competitive edge.
- The architecture is ideal for diverse real-world applications with complex data needs.
- MongoDB's approach simplifies development, making it attractive to a wide range of developers.
- The advantages over relational databases are evident in terms of both efficiency and scalability.
- MongoDB's design choices position it as a leading modern database solution.

# MongoDB Use Cases

- MongoDB is applied across diverse technologies, including Internet of Things (IoT).
- Its adaptability makes it suitable for mobile applications and personalized user experiences.
- Real-time analysis benefits from MongoDB's efficient data retrieval capabilities.
- Catalog management and content management are additional areas where MongoDB shines.
- MongoDB's broad range of use cases demonstrates its flexibility and applicability.

# MongoDB Customers

- eBay employs MongoDB for cloud management, search suggestions, and more.
- Shutterfly switched from Oracle to MongoDB for its 6 billion images.
- Aadhar, India's ID project, uses MongoDB for the world's largest biometric database.
- Electronic Arts relies on MongoDB for FIFA Online 3, their online multiplayer game.
- https://www.mongodb.com/who-uses-mongodb