VNUHCM-University of Science
Faculty of Information Technology

# Introduction to Big Data

MongoDB®

Group 01

VNUHCM-University of Science
Faculty of Information Technology

# Members

21127267 Phan Van Ba Hai

21127406 Tran Dinh Quang

21127556 Do Quoc Tri

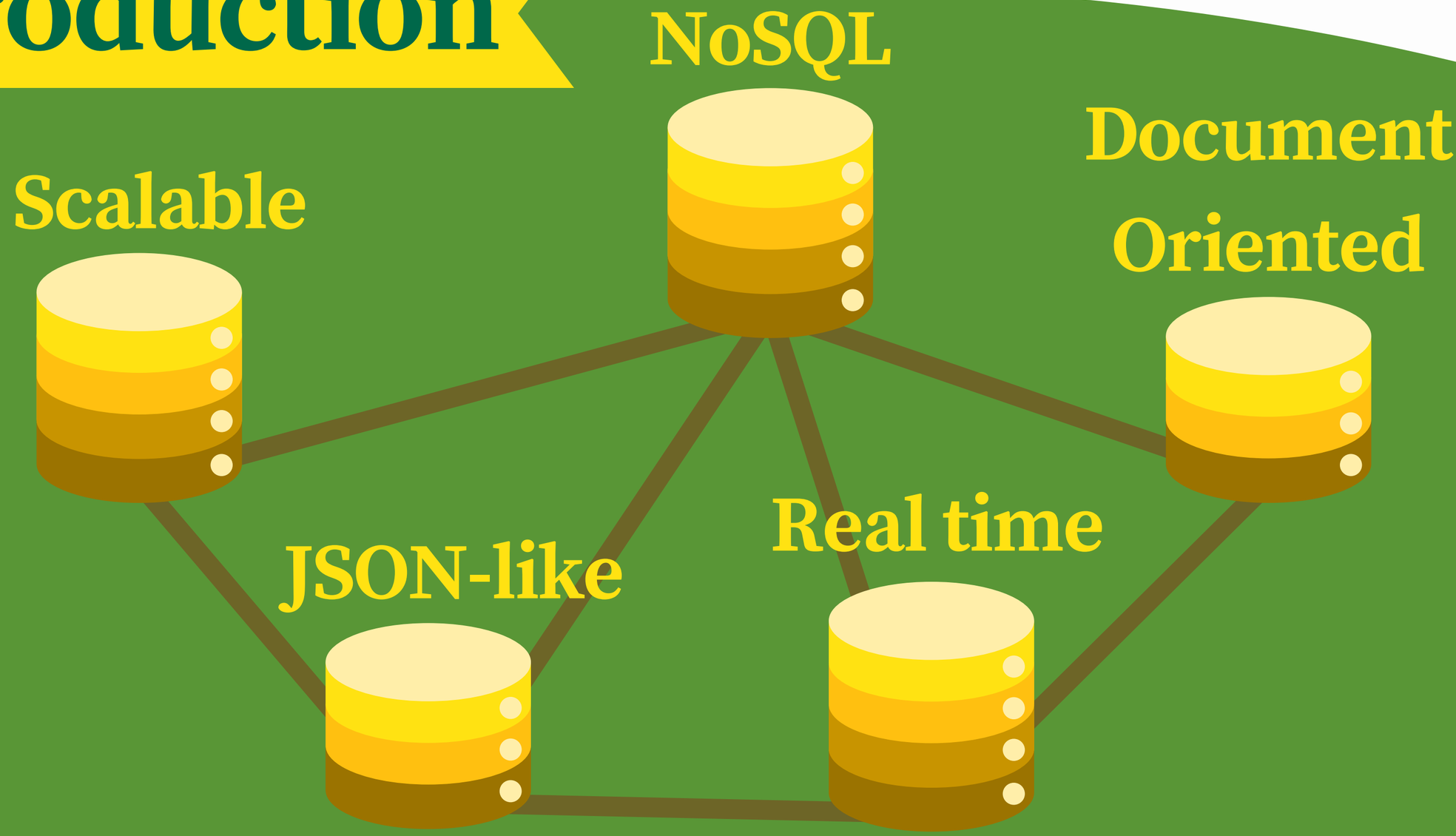
21127657 Nguyen Khanh Nhan

# Timeline

1. Introduction
   - History
   - Popularity
2. Key features
   - Storange Mechanism
   - Sharding & Scaling
   - Query API
   - High Availability
3. Major Component
   - Storing in DB
   - Indexes
   - Replication
   - Sharding
4. Major Functionality
5. Performance Comparison

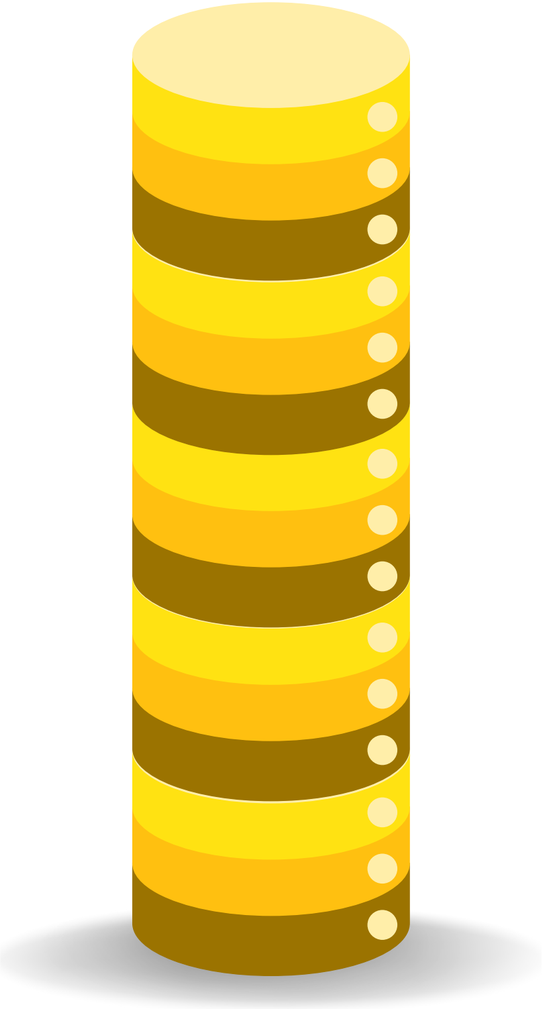
VNUHCM-University of Science
Faculty of Information Technology

# Introduction



# History

- **Initial Release (February 2009):** MongoDB introduced, pioneering document-oriented database management system.
- **Version 3.0 (March 2015):** WiredTiger storage engine default, enhancing performance and concurrency, with document-level locking, advancing scalability.
- **MongoDB Atlas (June 2016):** Fully managed cloud database service launched, simplifying administration tasks, enabling scalable deployment in the cloud.
- **Version 4.0 (June 2018):** Multi-document ACID transactions introduced, fulfilling community demand, expanding MongoDB's applicability in mission-critical environments.



# Popularity

- Technology
- Health care
- Education
- Finance
- E-commerse

...

SEGA® Forbes

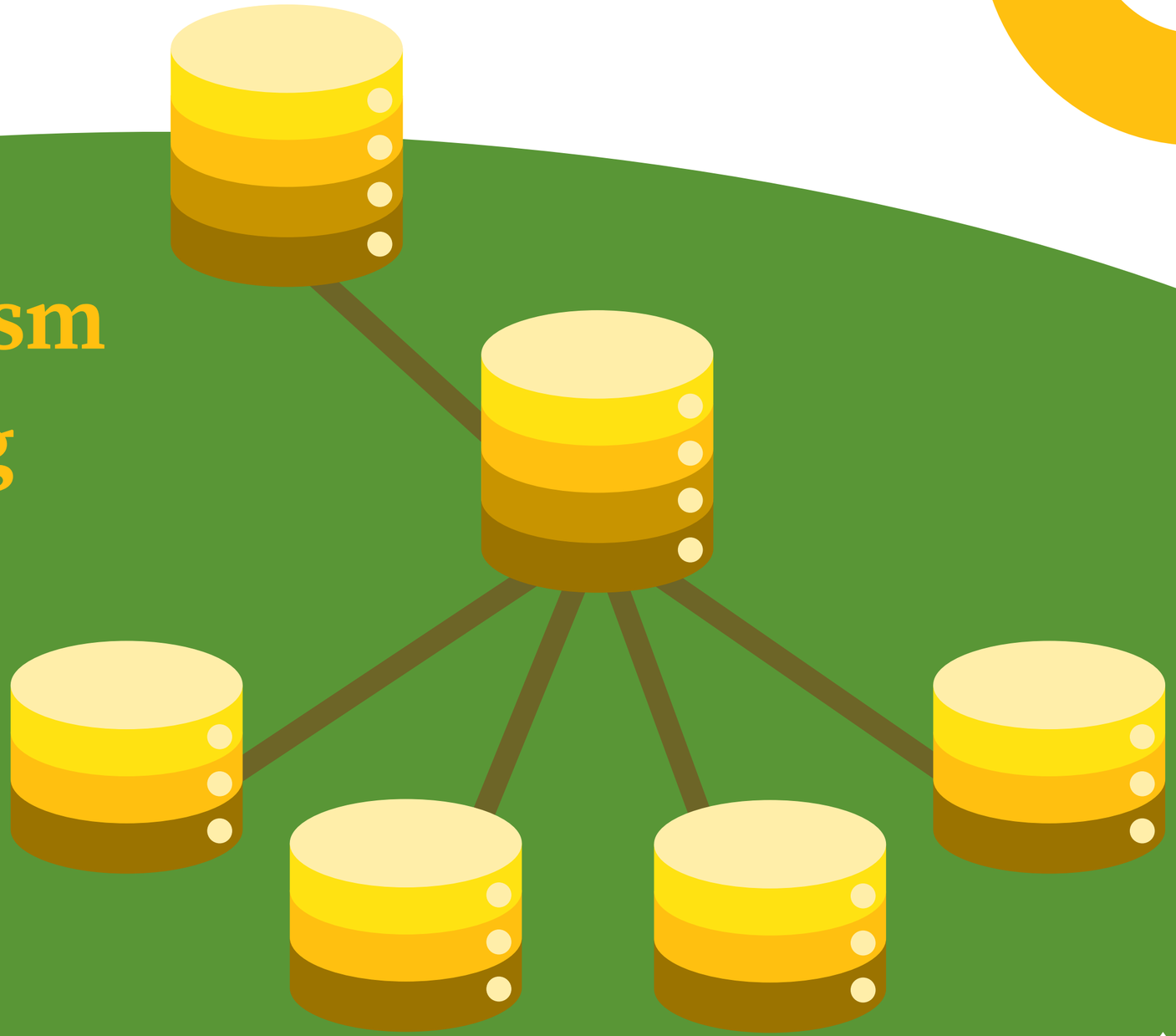
ebay Google

CISCO™ BOSCH

VNUHCM-University of Science
Faculty of Information Technology

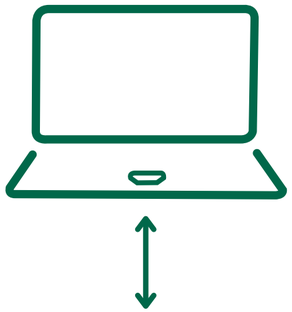
# Key features

- Storange Mechanism
- Sharding & Scaling
- High Performance
- High Available
- Query API



# Storage Mechanism

## Application



Primary

Secondary

Secondary

## Database

Collection

Collection

Collection

Documents

## Document

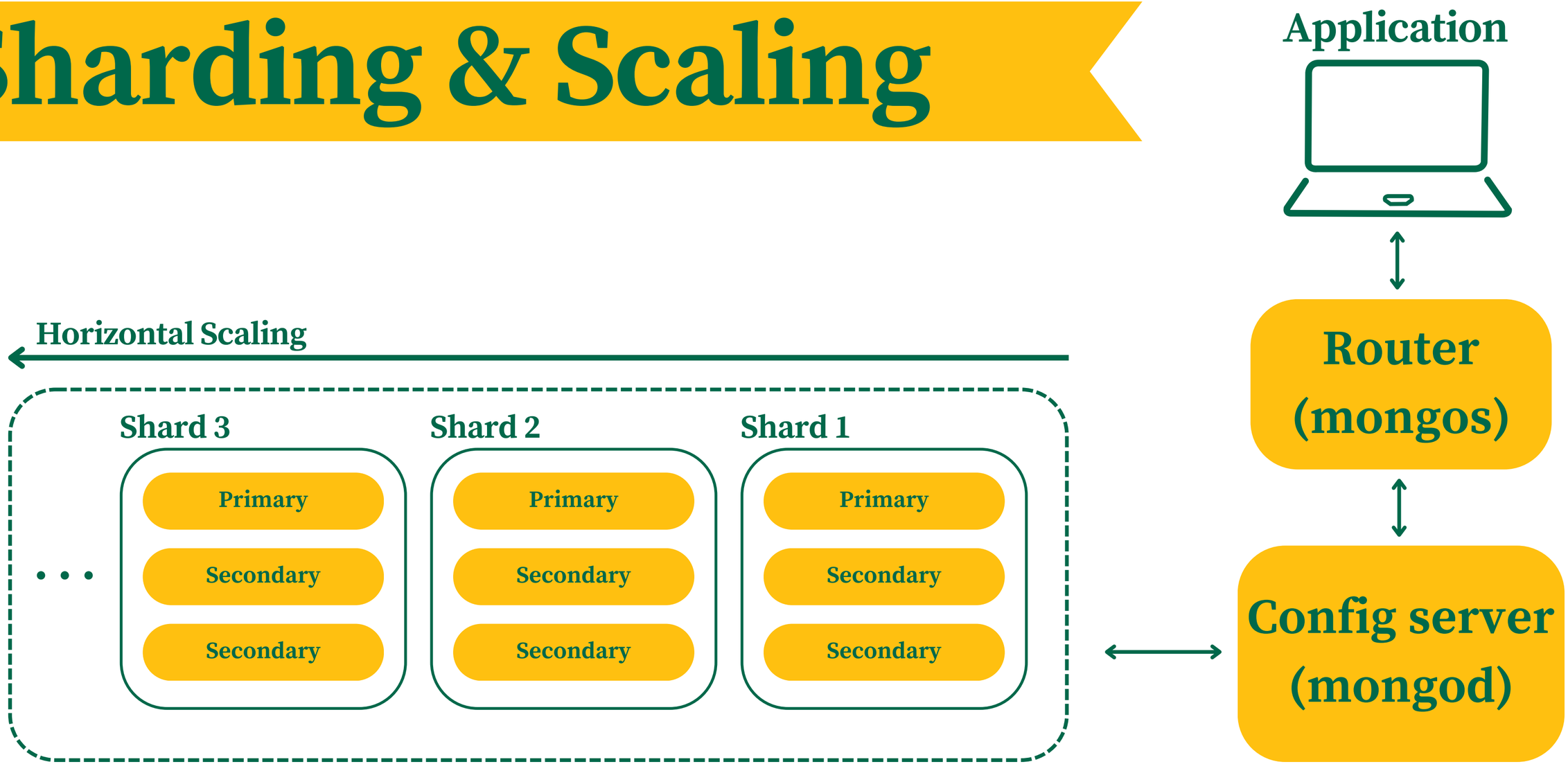
**BSON**

{

**'key': value**

'name': 'Nguyen A',
'birth': Date('2003-01-01'),
'id': 21127000
}

# Sharding & Scaling

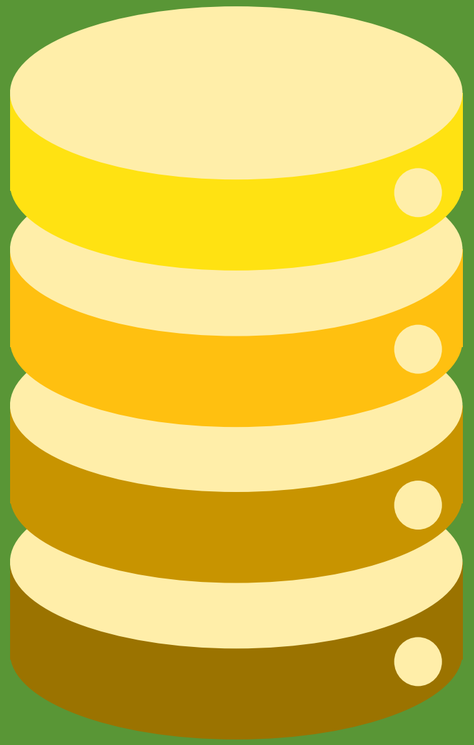


# Query API

- **CRUD Operations**
  - **Create**
  - **Read**
  - **Update**
  - **Delete**
- **Aggregation Pipelines**


- **Uses**
  - Adhoc queries (mongosh, Compass, VS Code, or MongoDB drivers).
  - Data transformations (aggregation pipelines).
  - Document joins across collections.
  - Graph and geospatial queries.
  - Full-text search.
  - Indexing for query performance.
  - Time series analysis.

# Major components



- Storing in mongoDB
- Indexing
- Replication
- Sharding

# Storing in mongoDB

## BSON

```
{
‘key’: value,
‘name’: ‘Nguyen A’,
‘id’: 21127000,
‘contact’: {
    ‘phone’: ‘012345678’
    ‘email’: ‘na@email.com’
            }
}
```

## Binary jSON

- ‘key’: value
- Additional Datatype
- Performance Optimize
- Nested Documents

# Indexing

| Class | Time |
|---|---|
| 21KHMT | 7:50 |
| 21MMT | 13:30 |
| 21HTTT | 7:30 |
| ... | ... |

| ID | Birth |
|---|---|
| 123 | 09/11 |
| 456 | 15/03 |
| 789 | 27/09 |
| ... | ... |

**Search for student who have ‘Score ’ > 5?**

- With out indexing, we must go through all documents.
- By using indexes, we just need to search on document with attribute “Score”.

| Name | ID |
|---|---|
| Pham D | 123 |
| Ly E | 456 |
| Do F | 789 |
| ... | ... |

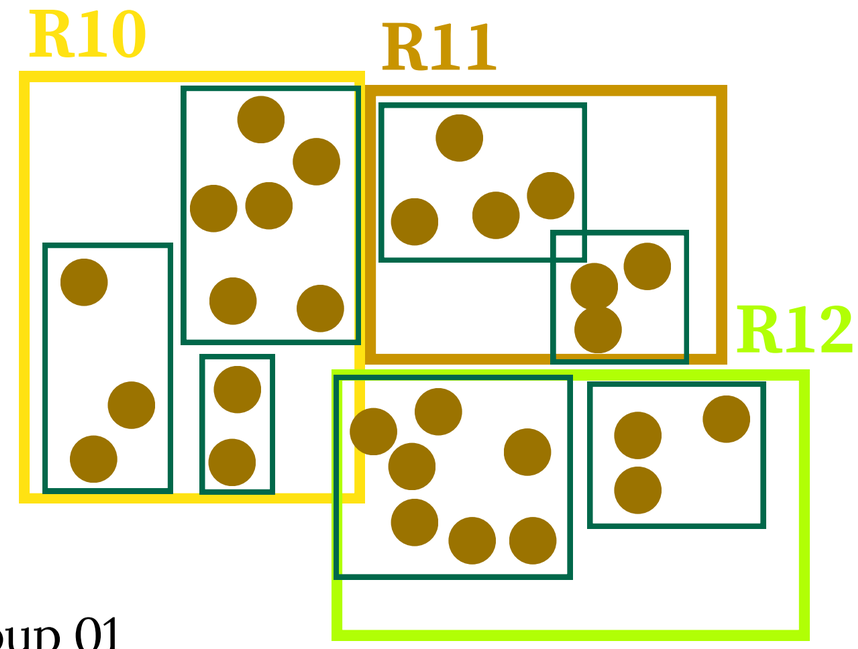
| Name | Score |
|---|---|
| Nguyen A | 4.5 |
| Phan B | 9 |
| Tran C | 7.5 |
| ... | ... |

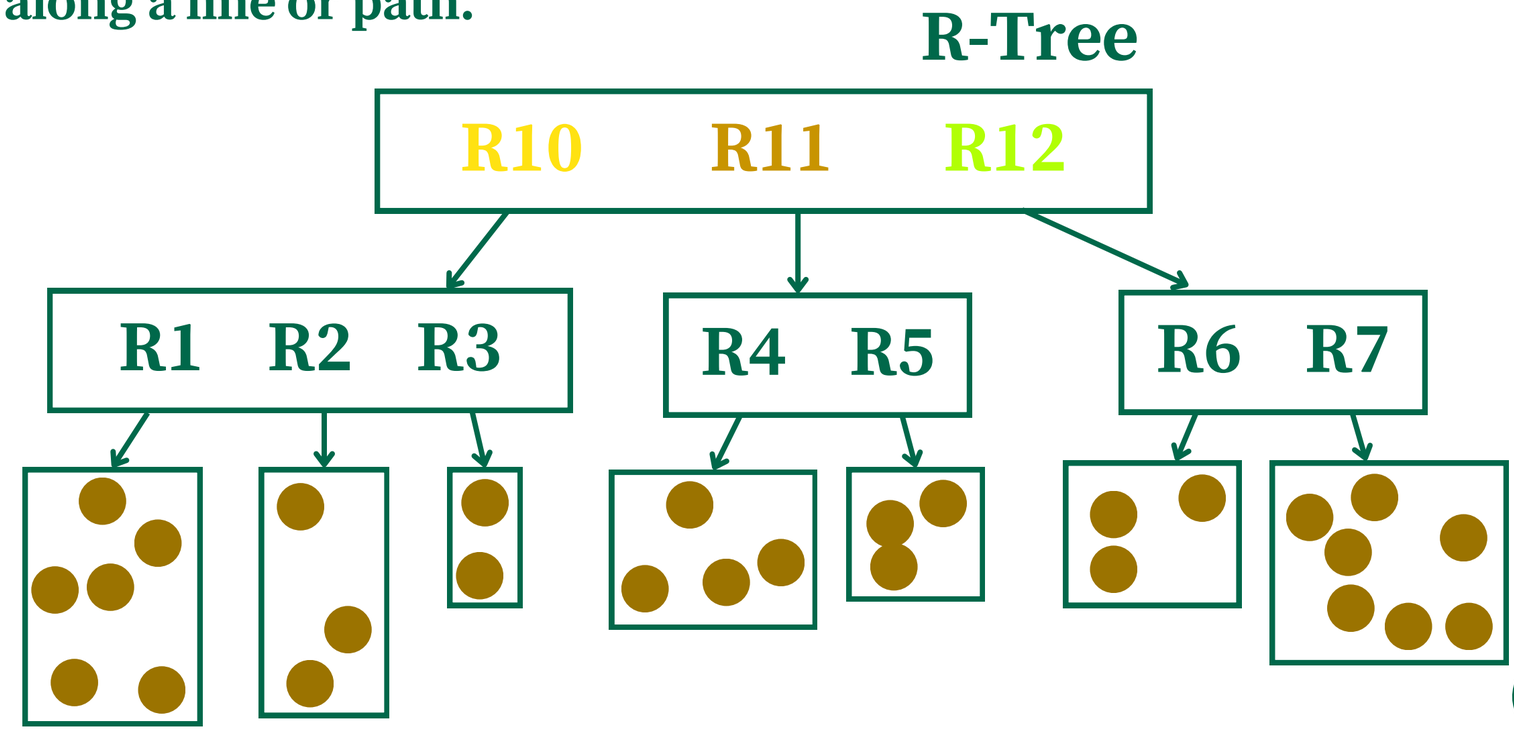
# Type of Indexing

- **Single Field Index:**
  - Indexes on a single field.
  - Efficient for queries filtering on that field.
  - Improves query performance.
- **Compound Index:**
  - Indexes on multiple fields.
  - Useful for queries involving multiple fields.
  - Order of fields in the index impacts query optimization.
- **Multikey Index:**
  - Indexes on arrays.
  - Facilitates queries on array elements.
  - Enables efficient querying of nested data.
- **Text Index:**
  - Specialized index for text search.
  - Supports natural language queries.
  - Suitable for text-heavy applications.
- **Geospatial Index:**
  - Indexes on geometrical data.
  - Enables efficient spatial queries.
  - Useful for location-based applications.
- **Hashed Index:**
  - Indexes using a hash of the field's value.
  - Distributes keys uniformly across shards.
  - Enhances performance for equality matches.

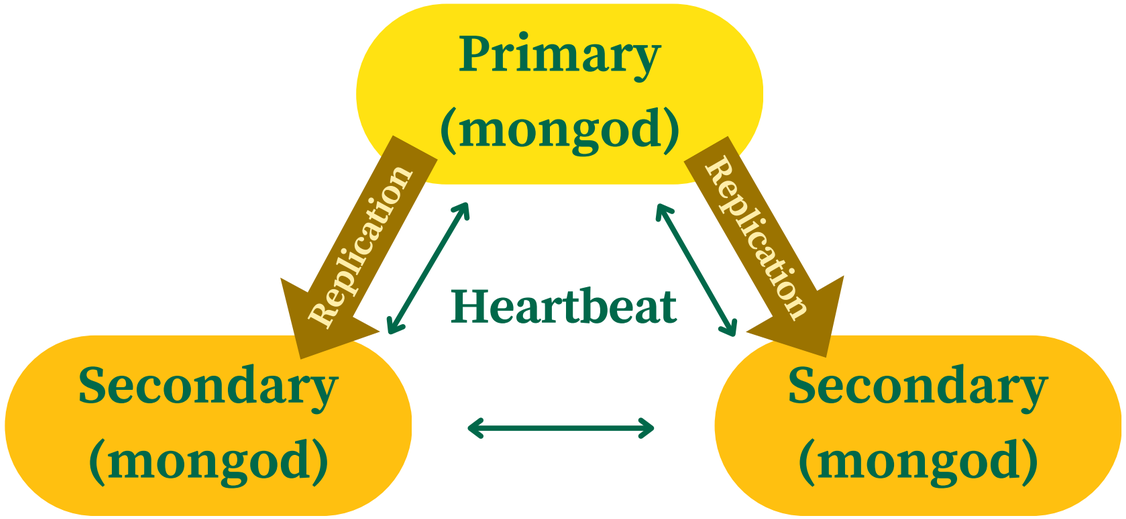
# Geospatial Indexing

- Near Queries: Finding documents near a specific point or geographical area.
- Within Queries: Finding documents within a specific geographical area, such as a rectangle or polygon.
- Along Queries: Finding documents along a line or path.



# Replication




**Replica set:** group of **mongod** instances that maintain the same data set.

**Primary Node (Primary Replica):**

- The main source for all write operations.
- The only node accepting write requests.
- Where all data modifications begin and are implemented initially.

**Secondary Nodes (Secondary Replica):**

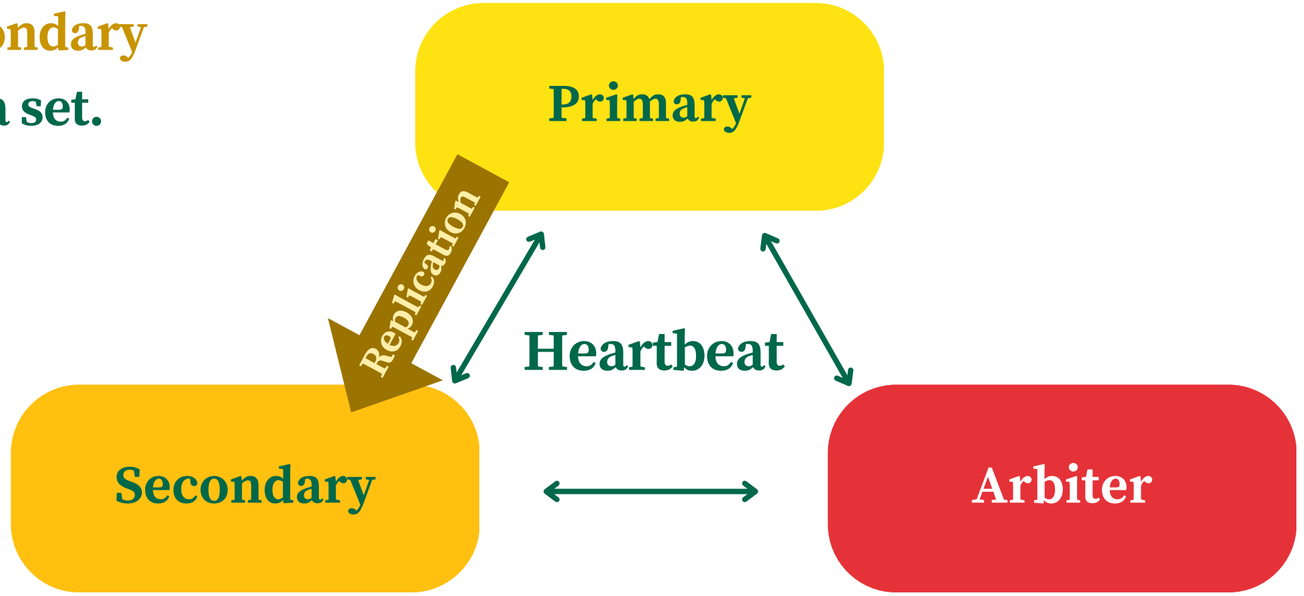
- Duplicate data from the primary node.
- Useful for dispersing read workloads and load balancing.
- Read-only and mostly utilized for read activities.

# Replication

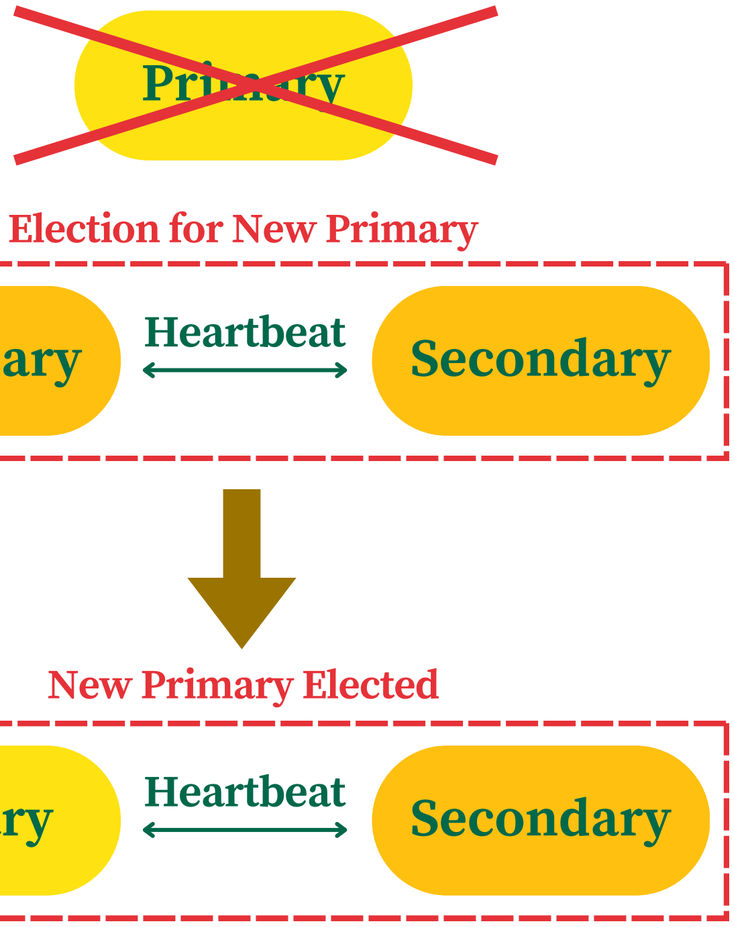
**Because of cost constraints, Adding another secondary is not possible, an arbiter can be added to a replica set.**

**Arbiter:**

- Participates in elections.
- Does not store data.
- Cost-effective option without providing data redundancy.



# Automatic Failover




When the **primary node** does **not communicate** with other members **longer than** the **electionTimeoutMillis** (default 10 seconds), An **Election** occurs!

**During the election:**

- Write operations cannot be processed.
- Read queries can still be served by secondaries.

Once a new primary is elected, normal operations resume.

# Sharding

**Shards:**

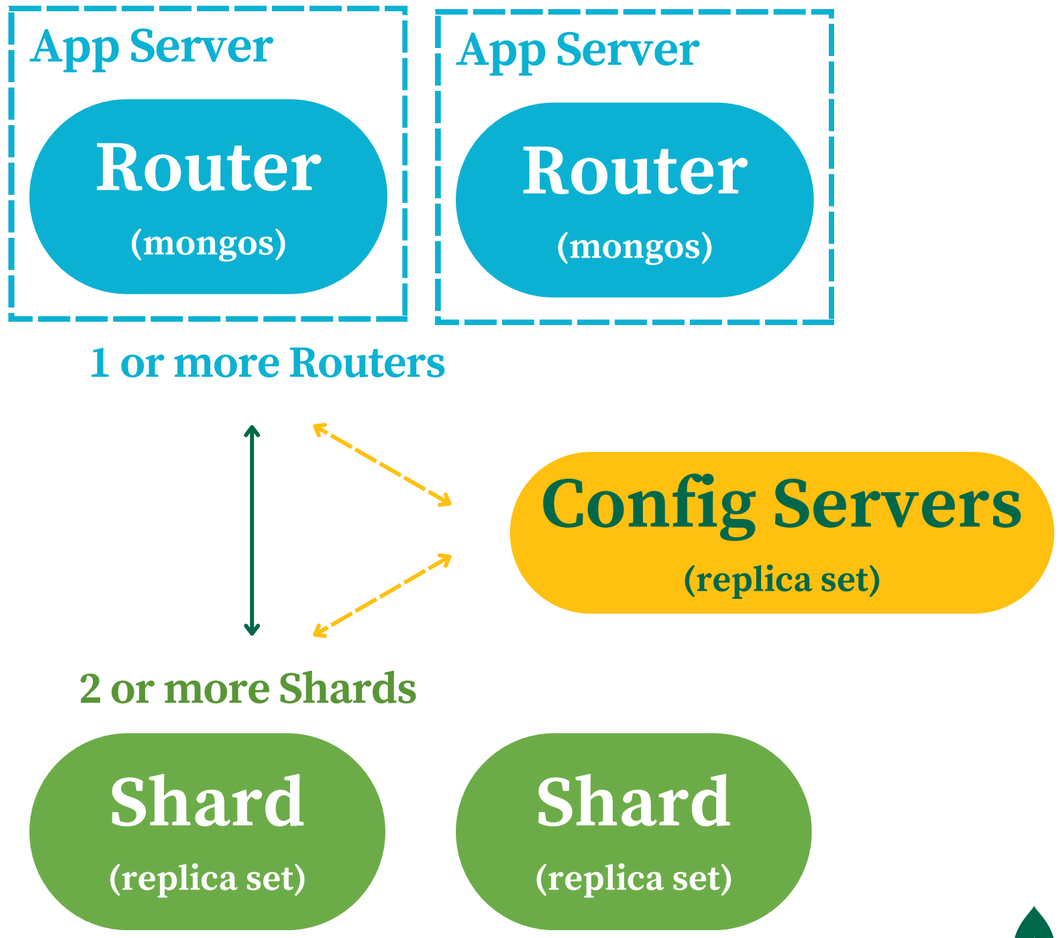
- Each shard contains a subset of the sharded data.
- Each shard must be deployed as a replica set.

**Mongos:**

- Query router.
- Interface between client applications and the sharded cluster.
- Support hedged reads to minimize latencies.

**Config servers:**

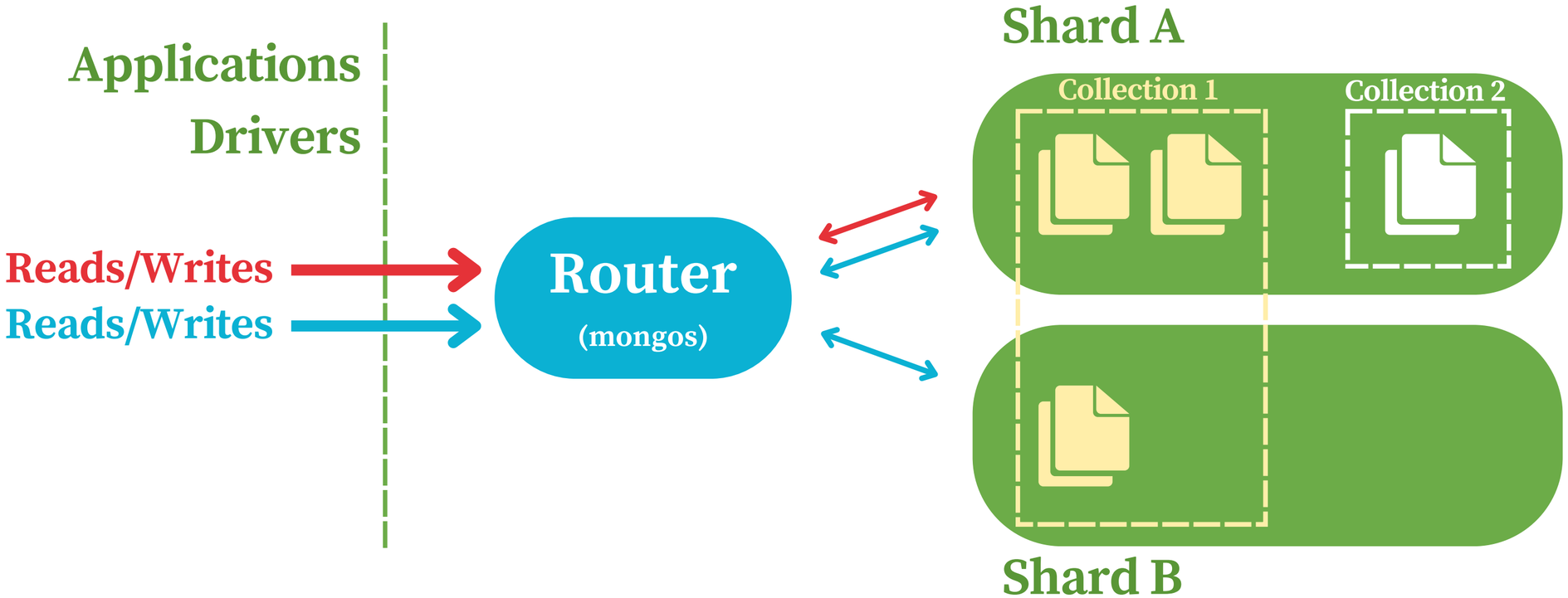
- Store meta data and configuration settings for the cluster.
- Must be deployed as a replica set.



# Connect to a Sharded Cluster

Clients must **connect to a mongos router to interact** with any collection in the sharded cluster (sharded and unsharded collections).

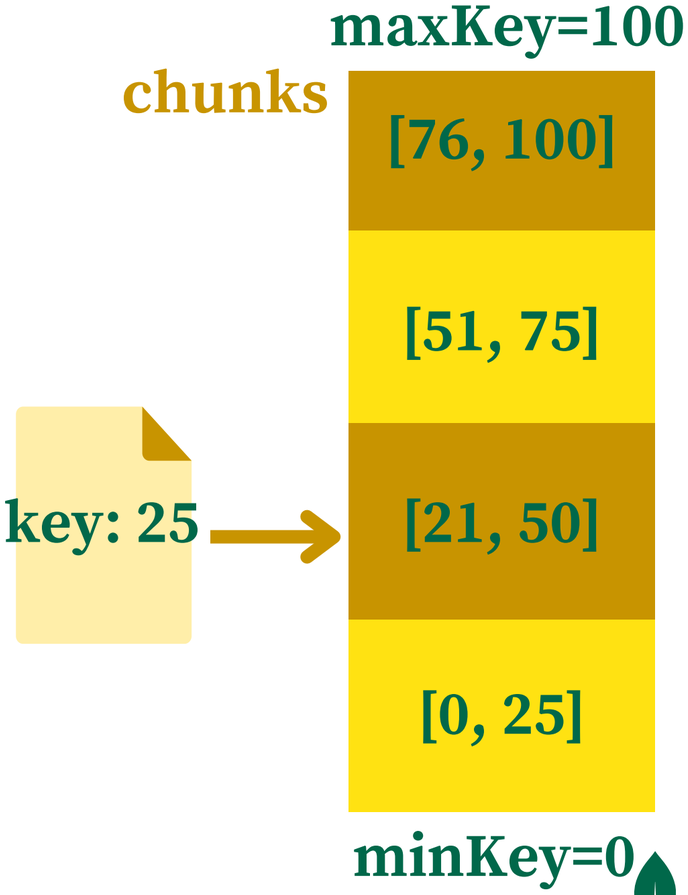
Clients **should never connect to a single shard** in order to perform **read** or **write** operations.



# Ranged vs Hashed Sharding

**Ranged Sharding:**

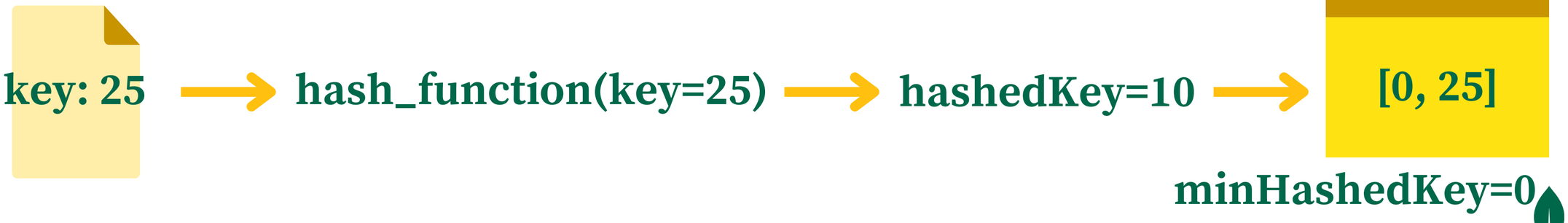
- **Method:**
  - Divides data into ranges based on the shard key values.
  - Assigns each chunk a range based on the shard key values.
- **Advantage:**
  - Facilitates targeted operations as shard keys close in value reside on the same chunk.
- **Disadvantage:**
  - Efficiency depends on well-chosen shard keys; poorly chosen keys can lead to uneven data distribution and performance issues.



# Ranged vs Hashed Sharding

**Hashed Sharding:**

- **Method:**
  - Computes a hash of the shard key field's value.
  - Assigns each chunk a range based on the hashed shard key values.
- **Advantage:**
  - Ensures even data distribution, particularly with monotonic shard key changes.
- **Disadvantage:**
  - May lead to cluster-wide broadcast operations for range-based queries.

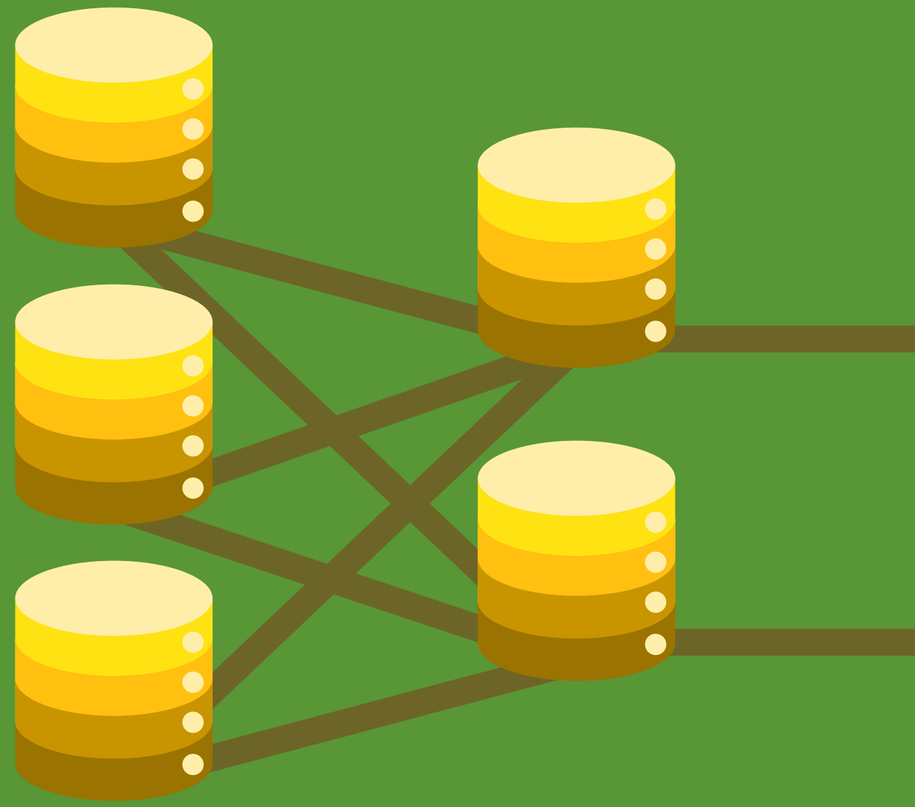


# Advantage of Sharding

- **Distributed Reads/Writes:**
  - Distribute the read and write workload.
  - Parallel process on a subset of cluster operations.
  - Targeted operations are more efficient than broadcasting to every shard.
- **Enhanced Storage Capacity and Fault tolerance:**
  - Distributes data across shards in the cluster.
  - Fault tolerance, each shard contains a subset of the total cluster data.
  - Horizontally Scalable.
- **Improved High Availability:**
  - Shards deployed as replica sets ensure high availability.
  - Even with some shards unavailable, the cluster can perform partial reads and writes.

# Major Functionality

- CRUD
- Aggregation Framework
- Transaction
- Query Optimization



# Major Functionality

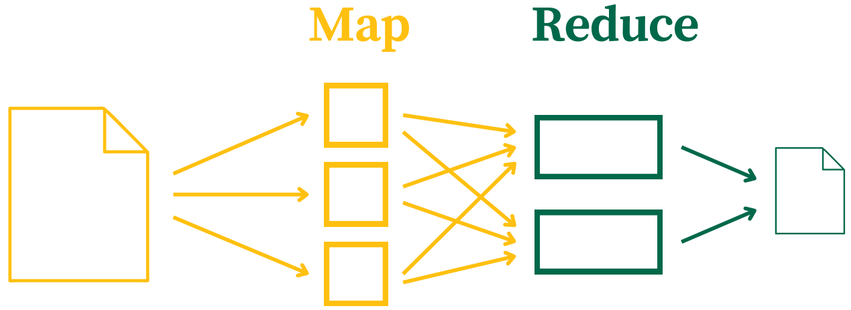
- **CRUD Operations**

  Create  Read

  Update  Delete

- **Aggregation Pipelines**


- **MapReduce**




- **Transactions**

VNUHCM-University of Science
Faculty of Information Technology

# Transaction & ACID

**Atomicity:**

- Transaction commands are treated as a **single** unit.
- They are **all** completed successfully or been rollbacked if **any** part of the transaction failed.

**Consistent with database constraints:**

- **Rules**
- **Constraints**
- **Triggers**

**Isolation:**

- All transactions run in an **isolated environment**.
- They **don't interfere** with each other.

**Durability:**

- Once the transaction completes and changes are written to the database, they are **persisted**.

# Query Optimize

**Explain() Method:**

- Used to analyze query performance.
- Provides insights into query execution, including query plan and index usage.
- Helps identify the need for additional indexes or optimizations.

**Projection:**

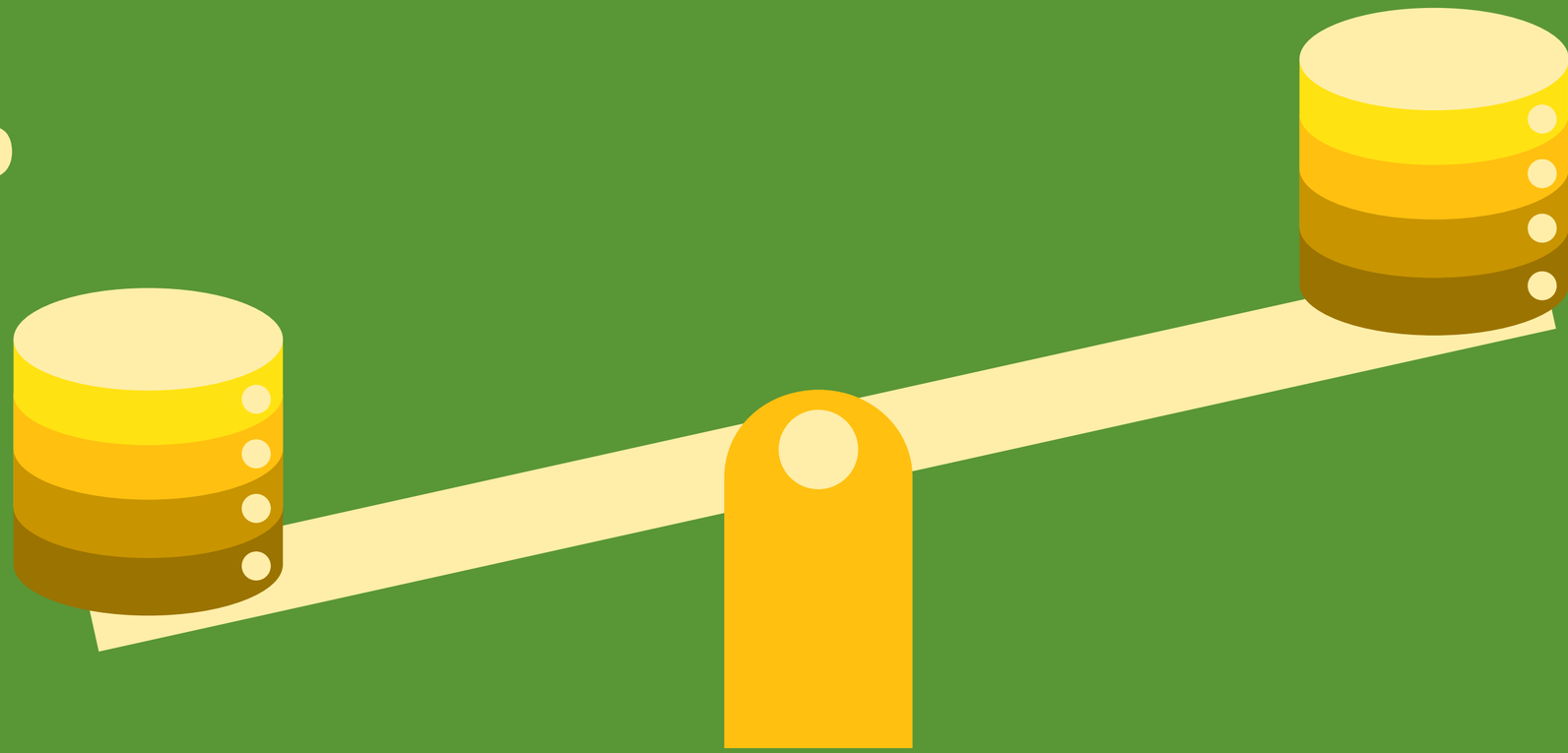
- Limits the fields returned in a query to only those needed.
- Reduces data transferred from the database to the application.
- Improves query performance and reduces memory usage.

**Pagination:**

- Utilizes limit() and skip() methods to retrieve a subset of data.
- Limits the number of documents returned and skips specified entries.
- Enhances query performance by minimizing data transfer.

# Comparison

- vs MySQL
- vs  Hadoop



# vs. Hadoop

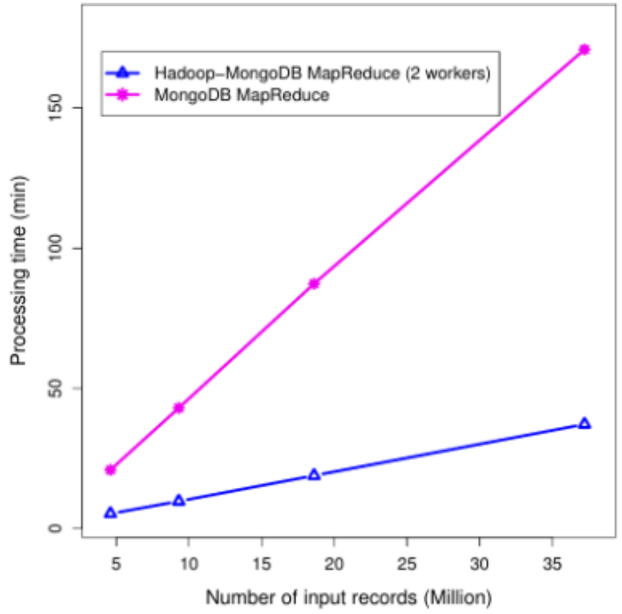
| Features | MongoDB | Hadoop |
|---|---|---|
| Data Storage | NoSQL, JSON-like, schema-less, sharding | HDFS, structured & unstructured data |
| Data Processing | Real-time, aggregation pipelines | Batch processing, MapReduce |
| Scalability | Horizontal scaling, sharding | Add nodes, petabytes of data |
| Query Language | MongoDB Query Language | MapReduce jobs, programming languages |
| Memory Handling | Memory optimization, caching | Disk-based, extensive disk I/O |
| Use Cases | Content management, real-time analytics | Big data analytics, data warehousing, log processing |

VNUHCM-University of Science
Faculty of Information Technology

# vs. Hadoop

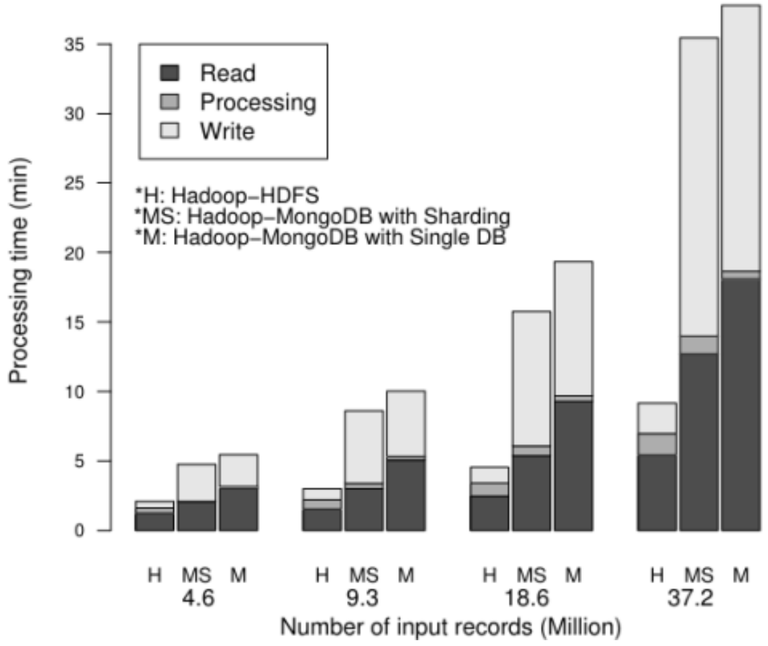
## MapReduce Performance

To obtain these results, the authors* ran MongoDB on a single server and used a 2-node Hadoop cluster. For this configuration, the mongo-hadoop plug-in provides roughly five times better performance.



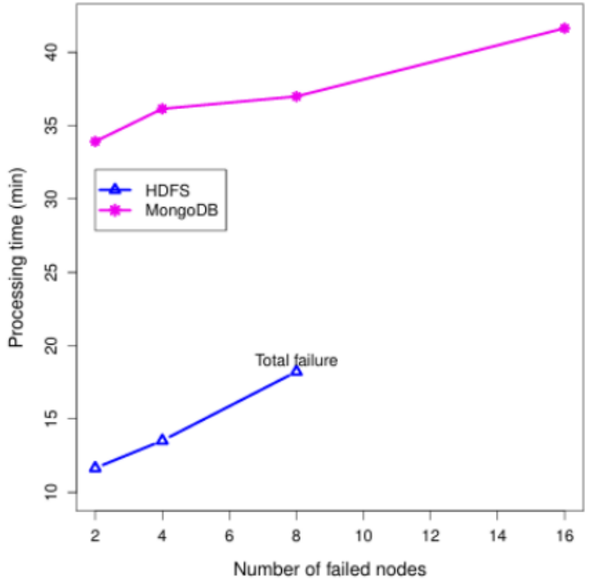

## Scalability

The figure shows how different setups perform as we increase the number of computers from 16 to 64 cores. The authors* compare using HDFS, one MongoDB server, and two MongoDB servers set up for sharing data.




## Fault tolerance

32-computer cluster dealing with 37 million records. Hadoop, if eight computers fail, too much data is lost, the job can't finish.
MongoDB, even if half of the computers fail, the job can still finish because it gets its data from MongoDB, not the computers. However, if the MongoDB server or one of the sharded servers fails, the job will fail too.






[*] E. Dede, M. Govindaraju, D. Gunter, R. Canon, and L. Ramakrishnan, “Performance Evaluation of a MongoDB and Hadoop Platform for Scientific Data Analysis,” Jun. 2013. (link)

# vs. MySQL

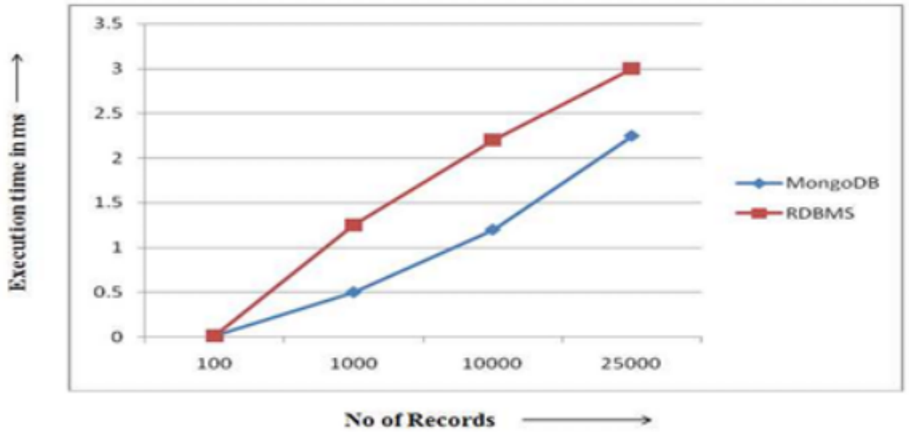
| Features | MongoDB (NoSQL) | MySQL (SQL) |
|---|---|---|
| Data Storage | NoSQL, JSON-like, schema-less, in-memory | RDBMS, tabular structure, predefined schemas |
| Data Processing | Real-time, aggregation pipelines | Batch processing, MapReduce |
| Scalability | Horizontal scaling, sharding | Vertical & horizontal scaling, replication, sharding |
| Schema | Schema flexibility, adaptability | Predefined schema, altering schema can be complex |
| Query Language | MongoDB Query Language | SQL |
| Use Cases | Content management, real-time analytics | E-commerce, content management, financial applications |

# vs. MySQL

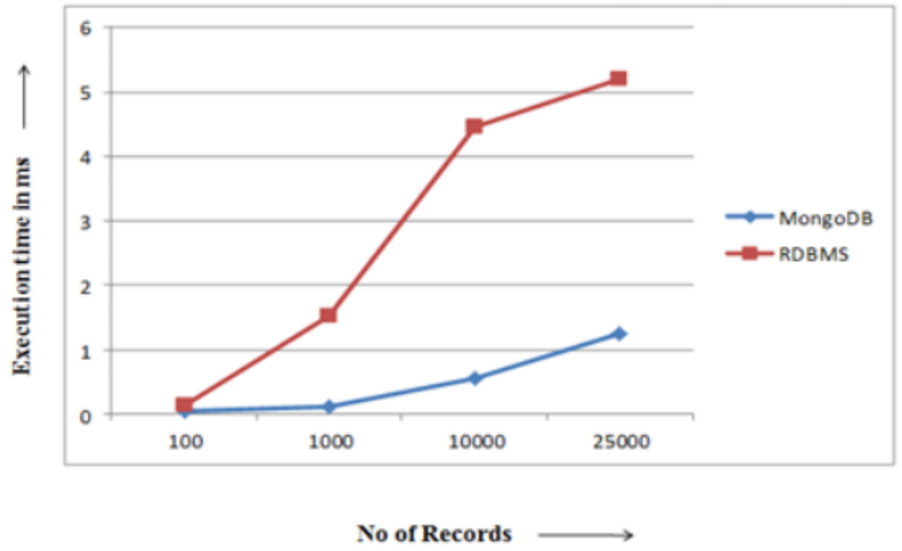
| Operation | Number of Records | Execution time (ms) | |
|---|---|---|---|
| | | MongoDB | MySQL |
| Insertion | 100 | 0.01 | 0.01 |
| | 1 000 | 0.5 | 1.25 |
| | 10 000 | 1.2 | 2.2 |
| | 25 000 | 2.25 | 3 |
| Search | 100 | 0.05 | 0.152 |
| | 1 000 | 0.12 | 1.52 |
| | 10 000 | 0.55 | 4.47 |
| | 25 000 | 1.25 | 5.21 |

[**] D. Damodaran B, S. Salim, and S. M. Vargese, “Performance Evaluation of MySQL and MongoDB Databases,” International Journal on Cybernetics & Informatics, vol. 5, no. 2, pp. 387–394, Apr. 2016, doi: https://doi.org/10.5121/ijci.2016.5241. (link)

## Insertion




## Search



# Conclusion

**Advantages:**

- Performance: RAM-based storage for quicker queries.
- Speed & Availability: Document-based structure, replication, gridFS.
- Setup Ease: Quick installation, JavaScript frameworks.
- Flexibility: Dynamic schema, supports non-structured data.
- Sharding: Automatic distribution for large datasets.
- Scalability: Horizontal scaling, expands storage.
- Documentation: Accurate, comprehensive support.
- Tech Support: Professional assistance, community forums.

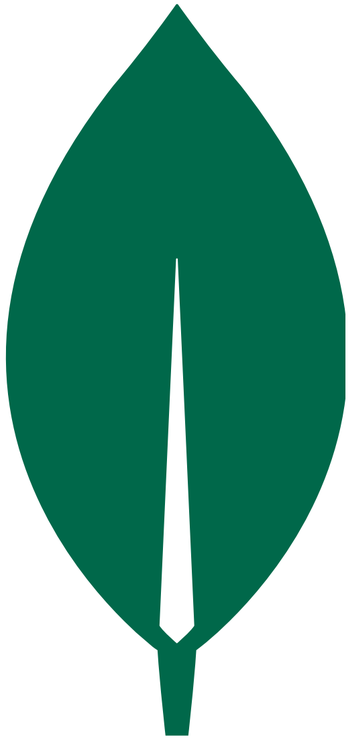
**Disadvantages:**

- Transactions: Limited support may affect reliability.
- Joins: Manual coding, potential complexity.
- Indexing: Errors may lead to low performance.
- Data Size & Nesting: Limits on size and nesting.
- Duplicates: Increased due to lack of relations.
- Memory Usage: High due to redundancy, no joins.

**Conclusion:**

- Popular for scalability, flexibility, and performance.
- Success stories from Forbes, Toyota highlight benefits.

# Q&A



# Thank you