MongoDB

Introduction

MongoDB is a NoSQL database that provides high performance, high availability, and easy scalability. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents, making it a popular choice for modern web applications.

Key Features

1. Schema-less Structure: No predefined schema, allowing flexibility.
2. Scalability: Supports horizontal scaling via sharding.
3. High Performance: Optimized for read and write operations.
4. Replication: Ensures high availability through replica sets.
5. Rich Query Language: Supports complex queries, aggregation, and indexing

Architecture

MongoDB stores data as documents in collections. The basic hierarchy is as follows:

1.Database → Contains multiple collections.

2.Collection → Groups multiple documents.

3.Document → A JSON-like object with key-value pairs.

Example Document:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "name": "John Doe",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "New York"
  }
}
```

MongoDB Advantages:

MongoDB is a powerful NoSQL database that offers several advantages over traditional relational databases.

Advantage	Description
Schema-less	Stores documents with flexible structures.
No Complex Joins	Queries data without requiring joins.
Scalability	Supports horizontal scaling with sharding.
High Performance	Uses internal memory for faster data access.
Rich Query Support	Allows dynamic queries with powerful filtering.
Replication	Ensures high availability with replica sets.

MongoDB Data Modeling:

MongoDB uses a flexible schema, allowing documents in the same collection to have different structures and field types. This provides greater adaptability compared to traditional relational databases.

Schema Design Considerations

1.Design the schema based on user requirements.

2.Store related data in the same document to minimize joins.

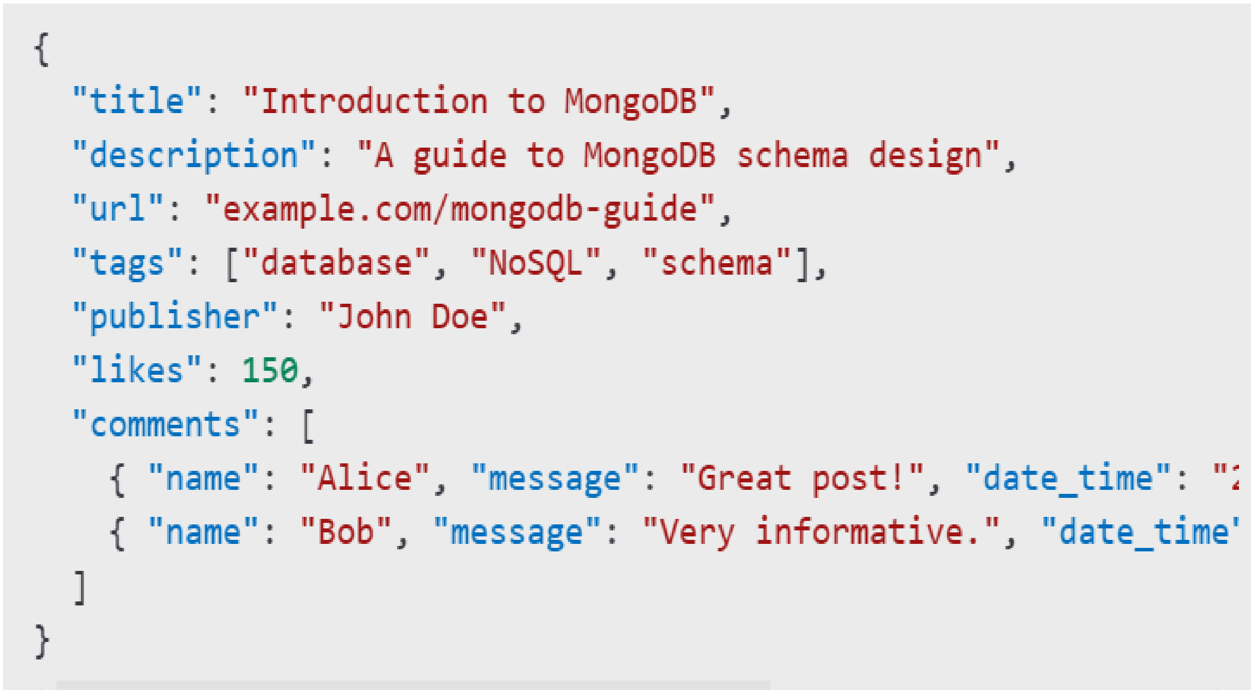
3.Duplicate data moderately since disk space is cheaper than computation time.

4.Perform joins at the write stage instead of during reads.

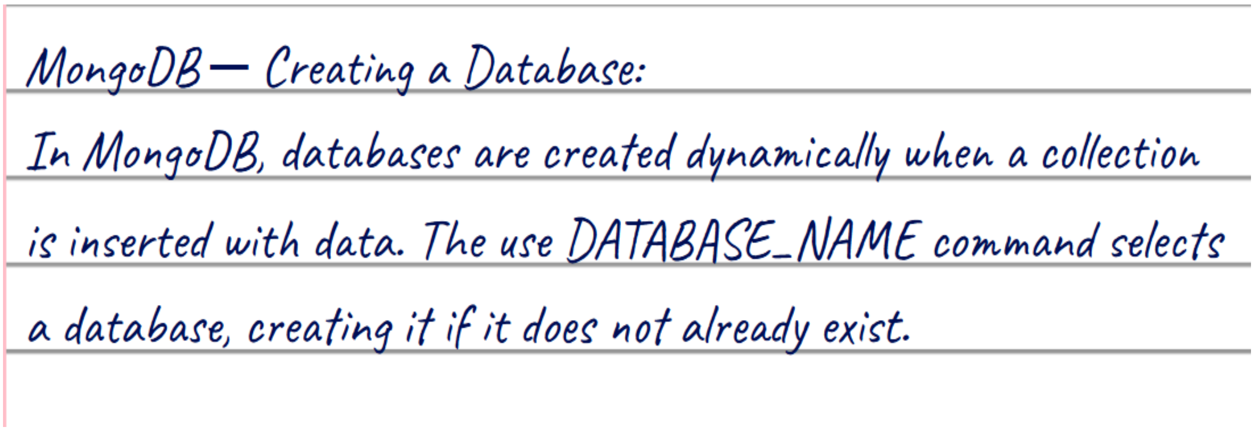
5.Optimize the schema for frequent queries.

6.Use complex aggregations when needed.

Example MongoDB document:



```json
{
  "title": "Introduction to MongoDB",
  "description": "A guide to MongoDB schema design",
  "url": "example.com/mongodb-guide",
  "tags": ["database", "NoSQL", "schema"],
  "publisher": "John Doe",
  "likes": 150,
  "comments": [
    { "name": "Alice", "message": "Great post!", "date_time": "
    { "name": "Bob", "message": "Very informative.", "date_time"
  ]
}
```



MongoDB — Creating a Database:

In MongoDB, databases are created dynamically when a collection is inserted with data. The use DATABASE_NAME command selects a database, creating it if it does not already exist.

Creating a Database

Syntax:

```
use DATABASE_NAME
```

Example:

```
> use mydb
switched to db mydb
```

To check the currently selected database:

```
> db
mydb
```

To list all databases:

```
> show dbs
local       0.78125GB
test        0.23012GB
```

Initially, a newly created database will not appear in the list until a document is inserted.

Inserting Data to Confirm Database Creation

```
> db.movie.insert({"name":"tutorials point"})
> show dbs
local       0.78125GB
mydb        0.23012GB
test        0.23012GB
```

Creating a Collection:

MongoDB collections are created automatically when data is inserted. However, collections can also be explicitly created using the createCollection() method.

Syntax:

```
db.createCollection(name, options)
```

- **name**: The name of the collection.
- **options**: (Optional) A document specifying collection settings.

Example:

```
> db.createCollection("mycollection")
{ "ok" : 1 }
```

To verify the collection:

```
> show collections
mycollection
system.indexes
```

Collection Options:

Option	Type	Description
capped	Boolean	If `true`, creates a fixed-size collection that overwrites old documents when full. Requires `size`.
autoIndexID	Boolean	If `true`, automatically creates an index on `_id` (default: `false`).
size	Number	Maximum size in bytes for a capped collection. Required if `capped` is `true`.
max	Number	Maximum number of documents allowed in a capped collection.

Dropping a Database:

The dropDatabase() Method

MongoDB provides the db.dropDatabase() command to delete the currently selected database.

Syntax:

```
db.dropDatabase()
```

Example:

```
> use mydb
switched to db mydb
> db.dropDatabase()
{ "dropped" : "mydb", "ok" : 1 }
```

To verify, list the databases again:

```
> show dbs
local       0.78125GB
test        0.23012GB
```

Important Considerations When Dropping a Database

1. Irreversible Action: Dropping a database permanently deletes all collections and data.
2. Current Database Only: The dropDatabase() command affects only the currently selected database.
3. Permissions Required: Users need administrative privileges.
4. System Databases Cannot Be Dropped: The local, admin, and config databases are protected.
5. No Confirmation: MongoDB does not prompt for confirmation before dropping a database, so use caution.

MongoDB Data Types:

MongoDB supports various data types for documents. Below is a list of commonly used types:

Data Type	Description
String	Stores text data.
Integer	Stores numeric data (int32 or int64).
Double	Stores floating-point numbers.
Boolean	Stores `true` or `false` values.
Object	Stores embedded documents.
Array	Stores multiple values in a list.
Timestamp	Stores timestamp values.
Null	Represents a null value.

Inserting Documents

To insert documents into a collection, use insertOne() or insertMany()

Syntax:

```
db.collection.insertOne(document)
db.collection.insertMany([document1, document2])
```

Example:

```
> db.users.insertOne({ name: "John", age: 25 })
{ "acknowledged" : true, "insertedId" : ObjectId("...") }
```

Querying Documents:

MongoDB provides find() to retrieve documents from a collection.

Syntax:

```
db.collection.find(query, projection)
```

Example:

```
> db.users.find({ age: 25 })
```

Updating Documents

To update documents, use updateOne(), updateMany(), or replaceOne().

Syntax:

```
db.collection.updateOne(filter, update)
db.collection.updateMany(filter, update)
```

Example:

```
> db.users.updateOne({ name: "John" }, { $set: { age: 26 } })
```

Deleting Documents

To delete documents, use deleteOne() or deleteMany().

Syntax:

```
db.collection.deleteOne(filter)
db.collection.deleteMany(filter)
```

Example:

```
> db.users.deleteOne({ name: "John" })
```

Indexing

Indexes improve query performance. The createIndex() method is used to create an index.

Syntax:

```
db.collection.createIndex({ field: order })
```

Example:

```
> db.users.createIndex({ age: 1 })
```

Aggregation

Aggregation operations process data records and return computed results.

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      total: { $sum: "$price" }
    }
  }
])
```

Replication

Replication ensures high availability by creating multiple copies of data across servers.

Commands:

```
rs.initiate()
rs.add("server2")
```

Backup and Restore

To back up a database, use mongodump, and to restore, use mongorestore.

Backup Command:

```
mongodump --db mydb --out /backup/
```

Restore Command:

```
mongorestore --db mydb /backup/mydb/
```

MongoDB Deployment

MongoDB can be deployed on a single server, as a replica set for high availability, or as a sharded cluster for scalability.

Standalone Deployment

A single MongoDB instance for development or small applications.

```
mongod --dbpath /data/db
```

Replica Set Deployment

For high availability, use replica sets:

```
mongod --replSet rs0 --port 27017 --dbpath /data/db1
```

MongoDB with Java:

MongoDB provides an official Java driver to interact with the database.

Connecting to MongoDB in Java

Maven Dependency

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.4.0</version>
</dependency>
```

Java Code to Connect

```
import com.mongodb.client.*;
import org.bson.Document;

public class MongoDBConnect {
    public static void main(String[] args) {
        MongoClient mongoClient = MongoClients.create("mongodb:/
        MongoDatabase database = mongoClient.getDatabase("mydb")
        System.out.println("Connected to database: " + database.
    }
}
```

MongoDB with PHP

MongoDB can be accessed using the PHP driver.

Installing PHP MongoDB Driver

```
sudo pecl install mongodb
```

Add this line to php.ini:

extension=mongodb.so

PHP Code to Connect

```
<?php
require 'vendor/autoload.php';

$client = new MongoDB\Client("mongodb://localhost:27017");
$db = $client->mydb;
echo "Connected to MongoDB";
?>
```