MongoDB

Creating a 1-week MongoDB eBook involves organizing the content into daily segments that cover key concepts and practical exercises. Here's a suggested outline for the eBook:

MongoDB in One Week

Day 1: Introduction to MongoDB

- **What is MongoDB?**
  - NoSQL databases vs. SQL databases
  - Document-oriented data model
- **Installation and Setup**
  - Installing MongoDB on various platforms (Windows, macOS, Linux)
  - Running MongoDB server and MongoDB shell
- **Basic Concepts**
  - Databases, Collections, Documents
  - JSON and BSON

Day 2: CRUD Operations

- **Create Operations**
  - Inserting documents
  - Bulk insert operations
- **Read Operations**
  - Querying documents with `find`
  - Query operators
  - Projection
- **Update Operations**
  - Updating documents
  - Update operators

    - Bulk update operations
- **Delete Operations**
    - Deleting documents
    - Bulk delete operations

Day 3: Indexing and Aggregation

- **Indexes**
    - Creating and managing indexes
    - Types of indexes (single field, compound, multi-key, text, geospatial)
    - Index performance considerations
- **Aggregation Framework**
    - Introduction to aggregation
    - Aggregation pipeline
    - Common stages (`$match`, `$group`, `$project`, `$sort`, `$limit`, `$skip`)

Day 4: Data Modeling and Schema Design

- **Data Modeling**
    - Document structure design
    - Embedding vs. referencing
    - Schema design patterns (one-to-one, one-to-many, many-to-many)
- **Schema Validation**
    - Defining validation rules
    - Using JSON Schema for validation

Day 5: Advanced Topics

- **Replication**
    - Setting up replica sets
    - Replica set configuration and management
    - Failover and recovery
- **Sharding**
    - Understanding sharding concepts
    - Setting up a sharded cluster
    - Shard keys and sharding strategies

Day 6: Performance Tuning and Optimization

- **Performance Tuning**
  - Analyzing performance with `explain`
  - Query optimization techniques
  - Index optimization
- **Monitoring and Profiling**
  - Using MongoDB tools for monitoring (Mongostat, Mongotop)
  - Database profiling and logs

Day 7: Backup, Security, and Administration

- **Backup and Restore**
  - Backup strategies
  - Using `mongodump` and `mongorestore`
- **Security**
  - Authentication and authorization
  - Role-based access control (RBAC)
  - Encryption
- **Administration**
  - Managing users and roles
  - Database and collection administration

Additional Resources

- **Official Documentation**
  - MongoDB Manual
  - MongoDB University courses
- **Community Resources**
  - MongoDB forums
  - Stack Overflow
- **Books and Articles**
  - Recommended reading list

This outline ensures that by the end of the week, readers will have a solid understanding of MongoDB's core concepts and practical skills for using it in real-world applications. Let me know if you need any specific sections expanded or additional topics included!

Day 1: Introduction to MongoDB

What is MongoDB?

MongoDB is a NoSQL database that uses a document-oriented data model. Unlike SQL databases, which store data in tables with rows and columns, MongoDB stores data in flexible, JSON-like documents. This allows for a more dynamic and schema-less data structure.

NoSQL vs. SQL Databases

- **SQL (Relational Databases):**
  - Structured data
  - Fixed schema
  - Tables, rows, columns
  - Examples: MySQL, PostgreSQL, Oracle
- **NoSQL (Non-relational Databases):**
  - Unstructured or semi-structured data
  - Dynamic schema
  - Collections, documents
  - Examples: MongoDB, Cassandra, Redis

Installation and Setup

Installing MongoDB

For detailed installation steps, refer to the MongoDB official documentation. Below are the general steps for installing MongoDB on different platforms.

Windows:

1. Download the MongoDB installer from the MongoDB website.
2. Run the installer and follow the setup instructions.
3. Add MongoDB's `bin` directory to the system's PATH.

macOS:

1. Install Homebrew if you haven't already.
2. Run the following command:

```
brew tap mongodb/brew
brew install mongodb-community
```

Linux:

3. Import the public key used by the package management system:

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
```

4. Create a list file for MongoDB:

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
```

5. Reload the local package database:

```
sudo apt-get update
```

6. Install the MongoDB packages:

```
sudo apt-get install -y mongodb-org
```

Running MongoDB Server and Shell

7. **Starting the MongoDB Server:**

```
mongod
```

This starts the MongoDB server on the default port 27017.

8. **Connecting to MongoDB Shell:**
Open another terminal window and run:

```
mongo
```

This connects to the MongoDB server and opens the interactive MongoDB shell.

Basic Concepts

Databases, Collections, Documents

- **Database:** A container for collections.
- **Collection:** A container for documents, equivalent to a table in SQL.
- **Document:** A record in a collection, equivalent to a row in SQL. Documents are stored in BSON format (Binary JSON).

Example: Creating a Database and Collection

```
use myDatabase  // Switch to 'myDatabase'. Creates the database if it doesn't exist.

db.createCollection("myCollection")  // Creates a collection named 'myCollection'.
```

Inserting a Document

```
db.myCollection.insertOne({
    name: "Alice",
    age: 25,
    address: {
        street: "123 Main St",
```

```
        city: "Wonderland",
    },
});
```

Querying a Document

```
db.myCollection
    .find({
        name: "Alice",
    })
    .pretty();
```

This code inserts a document into `myCollection` and then queries it by the `name` field.

JSON vs. BSON

- **JSON (JavaScript Object Notation):** Human-readable data format.
- **BSON (Binary JSON):** Binary representation of JSON-like documents, used internally by MongoDB for efficiency.

Summary of Day 1

Today, we covered the basics of MongoDB, including its installation and setup, and an introduction to its core concepts like databases, collections, and documents. You should now have MongoDB running on your machine and be able to interact with it using the MongoDB shell.

Day 2: CRUD Operations

Create Operations

Inserting Documents

In MongoDB, you can insert documents into a collection using the `insertOne` or `insertMany` methods.

Example: Inserting a Single Document

```
db.myCollection.insertOne({
    name: "Bob",
    age: 30,
    hobbies: ["reading", "hiking"],
});
```

Example: Inserting Multiple Documents

```
db.myCollection.insertMany([
    { name: "Charlie", age: 28, hobbies: ["swimming", "gaming"] },
    { name: "Dana", age: 22, hobbies: ["painting", "cycling"] },
]);
```

Read Operations

Querying Documents with `find`

The `find` method retrieves documents that match the query criteria. You can use query operators to refine your search.

Example: Querying Documents

```
db.myCollection.find({ age: { $gt: 25 } }).pretty();
```

This query retrieves all documents where the `age` field is greater than 25.

Projection

Projection allows you to specify which fields to include or exclude in the result set.

Example: Using Projection

```
db.myCollection
    .find({ age: { $gt: 25 } }, { name: 1, hobbies: 1, _id: 0 })
    .pretty();
```

This query retrieves documents with `age` greater than 25, but only includes the `name` and `hobbies` fields in the result.

Update Operations

Updating Documents

You can update documents using the `updateOne`, `updateMany`, or `replaceOne` methods. Update operators are used to modify specific fields.

Example: Updating a Single Document

```
db.myCollection.updateOne({ name: "Alice" }, { $set: { age: 26 } });
```

Example: Updating Multiple Documents

```
db.myCollection.updateMany({ age: { $lt: 25 } }, { $set: { status: "young" } });
```

Update Operators

- `$set`: Sets the value of a field.
- `$inc`: Increments the value of a field.
- `$push`: Adds an element to an array.

Example: Incrementing a Field

```
db.myCollection.updateOne({ name: "Bob" }, { $inc: { age: 1 } });
```

Delete Operations

Deleting Documents

You can delete documents using the `deleteOne` or `deleteMany` methods.

Example: Deleting a Single Document

```
db.myCollection.deleteOne({ name: "Charlie" });
```

Example: Deleting Multiple Documents

```
db.myCollection.deleteMany({ age: { $lt: 25 } });
```

Day 3: Indexing and Aggregation

Indexes

Indexes improve query performance by allowing the database to quickly locate documents. You can create and manage indexes using the `createIndex` method.

Example: Creating an Index

```
db.myCollection.createIndex({ name: 1 });
```

This creates an ascending index on the `name` field.

Types of Indexes

- **Single Field Index:** Indexes a single field.
- **Compound Index:** Indexes multiple fields.
- **Multi-key Index:** Indexes arrays.
- **Text Index:** Indexes text data for search.
- **Geospatial Index:** Indexes location data.

Aggregation Framework

The aggregation framework processes data and returns computed results. It uses an aggregation pipeline with stages.

Example: Aggregation Pipeline

```
db.myCollection.aggregate([
    { $match: { age: { $gt: 25 } } },
    { $group: { _id: "$status", averageAge: { $avg: "$age" } } },
    { $sort: { averageAge: -1 } },
]);
```

Common Stages

- `$match`: Filters documents.
- `$group`: Groups documents by a specified field.
- `$project`: Reshapes documents.
- `$sort`: Sorts documents.
- `$limit`: Limits the number of documents.
- `$skip`: Skips a specified number of documents.

Day 4: Data Modeling and Schema Design

Data Modeling

Document Structure Design

Designing the structure of documents is crucial for efficient data retrieval and storage. MongoDB's flexible schema allows for various design patterns.

Embedding vs. Referencing

- **Embedding:** Storing related data within a single document.
- **Referencing:** Storing related data in separate documents with references.

Example: Embedding

```
{
  name: "Alice",
  address: {
    street: "123 Main St",
    city: "Wonderland"
  }
}
```

Example: Referencing

```
{
  name: "Alice",
  addressId: ObjectId("60a6b9f8d69b1d6b4f4d9f7c")
}
```

Schema Design Patterns

- **One-to-One:** Embedding or referencing.
- **One-to-Many:** Embedding (for small related data) or referencing (for large related data).
- **Many-to-Many:** Using an intermediate collection for references.

Schema Validation

Defining Validation Rules

You can enforce data validation rules using JSON Schema.

Example: Defining Validation Rules

```
db.createCollection("users", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["name", "email"],
            properties: {
                name: {
                    bsonType: "string",
                    description: "must be a string and is required",
                },
```

```
                email: {
                    bsonType: "string",
                    pattern: "^.+@.+$",
                    description:
                        "must be a string and match the regular expression pattern",
                },
            },
        },
    },
});
```

Day 5: Advanced Topics

Replication

Replication ensures data redundancy and high availability by replicating data across multiple servers.

Setting Up Replica Sets

1. **Initialize the Replica Set:**

```
rs.initiate();
```

2. **Add Members to the Replica Set:**

```
rs.add("hostname:port");
```

3. **Check Replica Set Status:**

```
rs.status();
```

Sharding

Sharding distributes data across multiple servers to handle large datasets.

Setting Up a Sharded Cluster

4. **Start Config Servers:**

```
mongod --configsvr --replSet configReplSet --dbpath /data/configdb --port
27019
```

5. **Start Shard Servers:**

```
mongod --shardsvr --replSet shardReplSet --dbpath /data/sharddb --port 27018
```

6. **Start the Mongos Router:**

```
mongos --configdb configReplSet/hostname:27019 --port 27017
```

7. **Add Shards to the Cluster:**

```
sh.addShard("shardReplSet/hostname:27018");
```

Shard Keys and Sharding Strategies

- **Shard Key:** A field used to distribute documents across shards.

- **Hash Sharding:** Distributes documents based on a hashed value of the shard key.
- **Range Sharding:** Distributes documents based on a range of shard key values.

Day 6: Performance Tuning and Optimization

Performance Tuning

Analyzing Performance with `explain`

The `explain` method provides information on how MongoDB executes a query.

Example: Using `explain`

```
db.myCollection.find({ age: { $gt: 25 } }).explain("executionStats");
```

Query Optimization Techniques

- **Use Indexes:** Ensure queries use appropriate indexes.
- **Avoid Full Collection Scans:** Use selective queries to avoid scanning the entire collection.
- **Optimize Schema Design:** Structure documents to minimize the need for complex queries.

Monitoring and Profiling

Using MongoDB Tools for Monitoring

- **Mongostat:** Provides real-time statistics.

```
mongostat
```

- **Mongotop:** Shows read and write activity by collection.

```
mongotop
```

Database Profiling and Logs

- **Enable Profiling:**

```
db.setProfilingLevel(2);
```

- **View Profile Data:**

```
db.system.profile.find().pretty();
```

Day 7: Backup, Security, and Administration

Backup and Restore

Backup Strategies

Regular backups are essential for data recovery. Use `mongodump` and `mongorestore` for backups.

Example: Backup with `mongodump`

```
mongodump --out /backup/directory
```

Example: Restore with `mongorestore`

```
mongorestore /backup/directory
```

Security

Authentication and Authorization

Implement authentication to verify user identities and authorization to control access.

Enable Authentication:

1. **Start MongoDB with Authentication:**

   ```
   mongod --auth --port 27017
   ```

2. **Create an Admin User:**

```
use admin
db.createUser({
  user: "admin",
  pwd: "password",
  roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
})
```

Role-Based Access Control (RBAC)

Assign roles to users to control their access to database operations.

Example: Creating a User with Roles

```
use myDatabase
db.createUser({
  user: "dbUser",
  pwd: "password",
  roles: [{ role: "readWrite", db: "myDatabase" }]
})
```

Encryption

Encrypt data to protect it from unauthorized access.

Enable WiredTiger Encryption:

```
mongod --enableEncryption --encryptionKeyFile /path/to/keyfile
```

Administration

Managing Users and Roles

Create and manage users and their roles to control access to the database.

Example: Adding a Role to a User

```
db.grantRolesToUser("dbUser", [{ role: "dbAdmin", db: "myDatabase" }]);
```

Database and Collection Administration

Manage databases and collections to ensure smooth operation and performance.

Example: Renaming a Collection

```
db.oldCollection.renameCollection("newCollection");
```

Example: Dropping a Database

```
db.dropDatabase();
```

Additional Resources

Official Documentation

- MongoDB Manual
- MongoDB University

Community Resources

- MongoDB Forums
- Stack Overflow

Books and Articles

- **"MongoDB: The Definitive Guide"** by Kristina Chodorow
- **"MongoDB in Action"** by Kyle Banker

This concludes the detailed explanation and code examples for each day of the 1-week MongoDB eBook. Let me know if you need further customization or additional topics!