**Discussion 6.1 – Indexes**

Patrick Cuauro

Bellevue University

WEB 335-307O Introduction to NoSQL

Professor Krasso

June 27, 2023

## What are indexes?

Indexes are data structures that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. Indexes are used to improve the performance of queries by allowing MongoDB to quickly find the documents that match a query.

## Why are indexes used and how are they created in MongoDB?

Without indexes, MongoDB must scan every document in a collection to find the documents that match a query. This scan is called a collection scan, and it can be very slow, especially for large collections. Indexes allow MongoDB to skip over documents that do not match the query, which can significantly improve the performance of queries.

There are many different types of indexes in MongoDB, but the most common are single-field indexes and compound indexes. Single-field indexes are indexes on a single field, while compound indexes are indexes on multiple fields.

Indexes can be created on any field in a collection, but the most effective indexes are created on fields that are frequently used in queries. For example, if you have a collection of products and you frequently query the collection by product name, you will want to create an index on the product name field.

Indexes can also be created on fields that are used in aggregations. For example, if you frequently aggregate the products by price, you will want to create an index on the price field.

Indexes are a powerful tool that can be used to improve the performance of queries in MongoDB. However, indexes also have a cost, as they consume disk space and memory. Therefore, it is important to create indexes only on fields that are frequently used in queries.

Here are some of the benefits of using indexes in MongoDB:

* Improved query performance: Indexes allow MongoDB to quickly find the documents that match a query, which can significantly improve the performance of queries.
* Reduced disk space usage: Indexes only store a small portion of the data set, which can help to reduce disk space usage.
* Improved data integrity: Indexes can help to improve data integrity by preventing duplicate documents from being inserted into a collection.

Here are some of the drawbacks of using indexes in MongoDB:

* Index maintenance overhead: Indexes must be maintained as documents are added, updated, or deleted from a collection. This can add some overhead to the database.
* Index storage overhead: Indexes consume disk space and memory. This can be a concern for large collections or collections with many indexes.

Overall, indexes are a powerful tool that can be used to improve the performance of queries in MongoDB. However, it is important to use indexes wisely, as they can also have some drawbacks.

## What is index cardinality?

The type of query operation in the query shape affects the order of the fields used to construct the index. In general, the Performance Advisor ranks fields by their cardinality.

Cardinality: The measure of the number of elements within a set of values. For example, the set

|  |
| --- |
| A = { 2, 4, 6 } |

 contains 3 elements and has a cardinality of 3.

## What are capped collections? And, when should they be used (provide at least one example of a scenario when they should be used)?

“Normal” collections in MongoDB are created dynamically and automatically grow in size to fit additional data. MongoDB also supports a different type of collection, called a capped collection, which is created in advance and is fixed in size.

Having fixed-size collections brings up an interesting question: what happens when we try to insert into capped collection that is already full? The answer is that capped collections behave like circular queues: if we’re out of space, the oldest document will be deleted, and the new one will take its place. This means that capped collections automatically age out the oldest documents as new documents are inserted.

Here are some of the use cases for capped collections:

* Logging: Capped collections are often used to store log data. This is because log data is typically very large, and it is important to keep it in a fixed size. Capped collections allow you to do this without having to worry about the collection growing too large.
* Chat: Capped collections can also be used to store chat messages. This is because chat messages are typically short, and they are constantly being updated. Capped collections allow you to store chat messages in a way that is efficient and easy to query.
* Time series data: Capped collections can also be used to store time series data. This is because time series data is typically very large, and it is important to keep it in a fixed size. Capped collections allow you to do this without having to worry about the collection growing too large.

Here are some of the advantages of using capped collections:

* Fixed size: Capped collections have a fixed size, which makes them ideal for storing data that is constantly being updated. This is because you don't have to worry about the collection growing too large and running out of space.
* Efficient queries: Capped collections are efficient to query, as they only need to scan a small portion of the collection to find the desired documents. This is because capped collections are sorted by insertion order, which means that the documents are stored in a contiguous block on disk.
* Simple to manage: Capped collections are simple to manage, as they do not require any indexes. This is because the documents in a capped collection are already sorted by insertion order, which means that MongoDB can quickly find the desired documents without the need for an index.

Here are some of the disadvantages of using capped collections:

* Data loss: If a capped collection reaches its maximum size and there are no documents that can be overwritten, the oldest documents in the collection will be lost. This means that you need to be careful about how you manage capped collections, as you could lose data if you are not careful.
* Limited scalability: Capped collections are not as scalable as regular collections. This is because capped collections have a fixed size, which means that they cannot grow to accommodate more data.

## What are sparse indexes?

Sparse indexes are a type of index in MongoDB that only indexes documents that have the indexed field. This means that if a document does not have the indexed field, it will not be included in the index.

Sparse indexes are useful for fields that are not always present in all documents. For example, if you have a collection of documents that represent users, you might have a field called *email*. Not all users will have an email address, so if you create a regular index on the email field, all documents in the collection will be indexed, even those that do not have an email address. This can waste space and make queries slower.

With a sparse index, only documents that have an *email* address will be indexed. This means that the index will be smaller, and queries will be faster.

Here is an example of how to create a sparse index in MongoDB:

|  |
| --- |
| db.users.createIndex({ "email": 1 }, { sparse: true }); |

This will create an index on the *email* field of the *users* collection. Only documents that have an email address will be indexed.

Sparse indexes are a good way to improve the performance of queries on fields that are not always present in all documents. However, they should be used with caution, as they can make updates slower.

Sources:

<https://medium.com/@alok5633y/query-optimization-in-mongodb-and-flask-d3613c2c415f>

<https://www.mongodb.com/docs/atlas/performance-advisor/index-ranking/>