Indexing in MongoDB

Learn about the benefits of creating indexes to support your queries in MongoDB.

What is Indexing?

Imagine if, for every query in MongoDB, we had to search the entire database to find our desired result. At scale, this would be a huge expenditure of time, computing power, and money. Fortunately, there's a tool that can help make certain queries much more efficient - indexing. An **index** is a special data structure that stores a small portion of the collection's data set in an easy-to-traverse form.

Let's use a real-world example to understand how indexing in MongoDB works. Suppose in your global history class your assignment is to write an essay about Japan. Naturally, you'll want to find all references in your history textbook about the country. But how can you find all pages that mention Japan? One method would be to look meticulously through every page in the textbook. This method is tedious, exhausting, and a poor use of your time. Using an index, however, you could go to the letter "J" in the index, locate Japan, and proceed to the corresponding pages listed.

Using an index to look up all references of the word "Japan" is a much more efficient way to search compared to a complete scan of the book. Indexes in MongoDB seek to capture that same efficiency and optimize query performance. Queries that don't use indexes must parse every document in a collection to find the appropriate matches. For relatively small collections this is fine, but as a collection grows this can begin to weigh down performance.

Multiple choice

Which of the following is a downside to querying an entire MongoDB collection without proper indexing?

- The query must run through every single document within the collection before returning a result.
- The query will stop after reaching one document that fits the filtering criteria.
- There are no downsides to querying before creating an index.
- The collection will be changed upon scanning through the collection.

Correct! This can put a considerable strain on computing power and bandwidth depending on the size of the collection.

The Types of Indexes in MongoDB

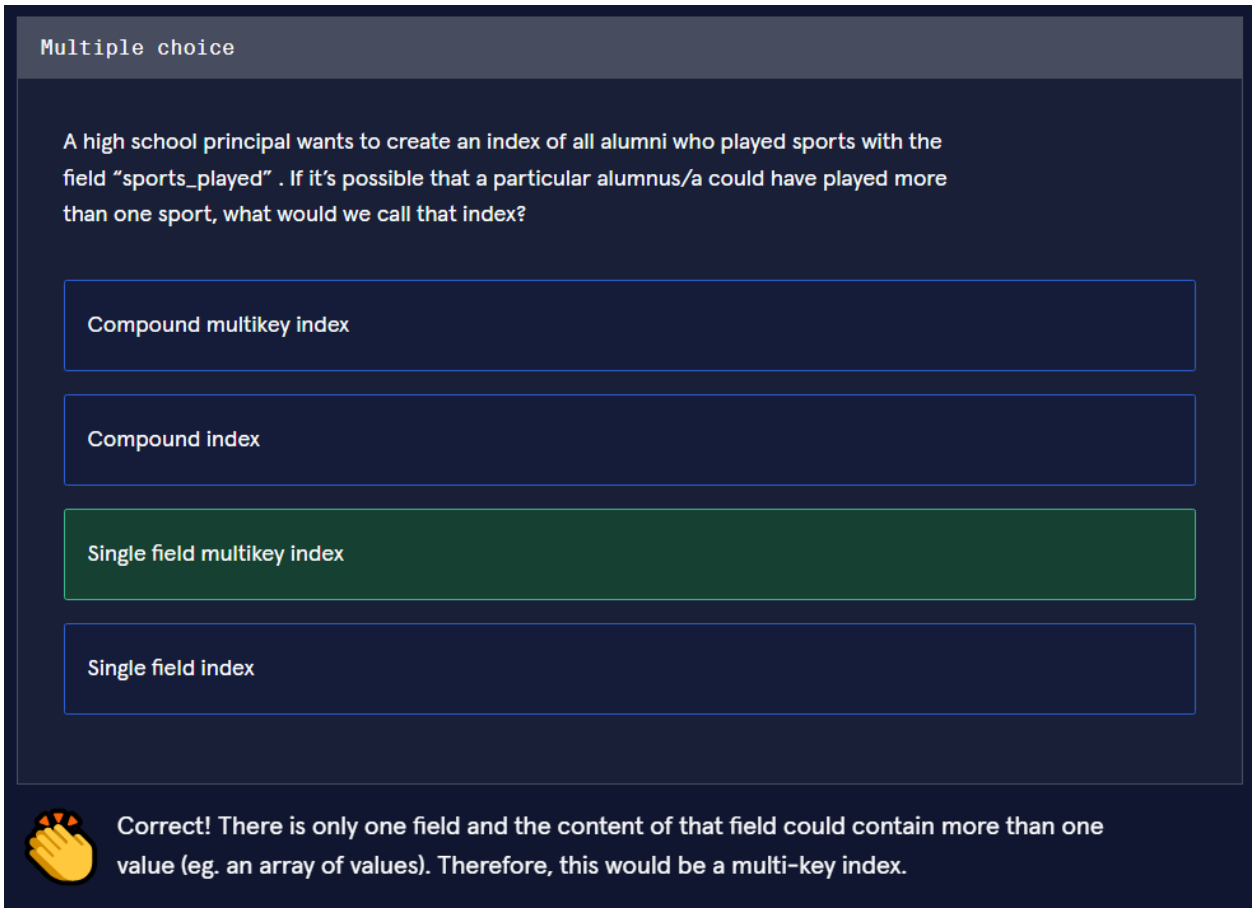
MongoDB supports several different types of indexes. You can, for instance, create an index that references only one field of a document - also known as a **single-field index**. If, for example, a high school principal wanted to organize a reunion for alumni who studied abroad in Argentina, they would want to run a query on all alumni who studied internationally, specifically in Argentina. Rather than query the entire alumni collection, they could capture a subset of this data by creating a single-field index on a field that's exclusive to these alumni, for example, `study_abroad_nation`. Let's take a look at what this single field index might look like:

The principal can now use this single field index to query specific study abroad countries, like Argentina. This index makes our query more economical because it allows us to scan a subset of data rather than every document in the collection. Furthermore, because indexes arrange data in ascending or descending order, our queries are able to more quickly locate matching documents, making them more optimized for speed.

You can also create indexes on multiple fields, called **compound indexes**, to support more specific queries. If that same high school principal wanted to organize these alumni based on the year they studied abroad, he or she could create a compound index that references the country where students studied abroad *and* when they studied abroad. The principal can now use this index to query specific countries, and, since

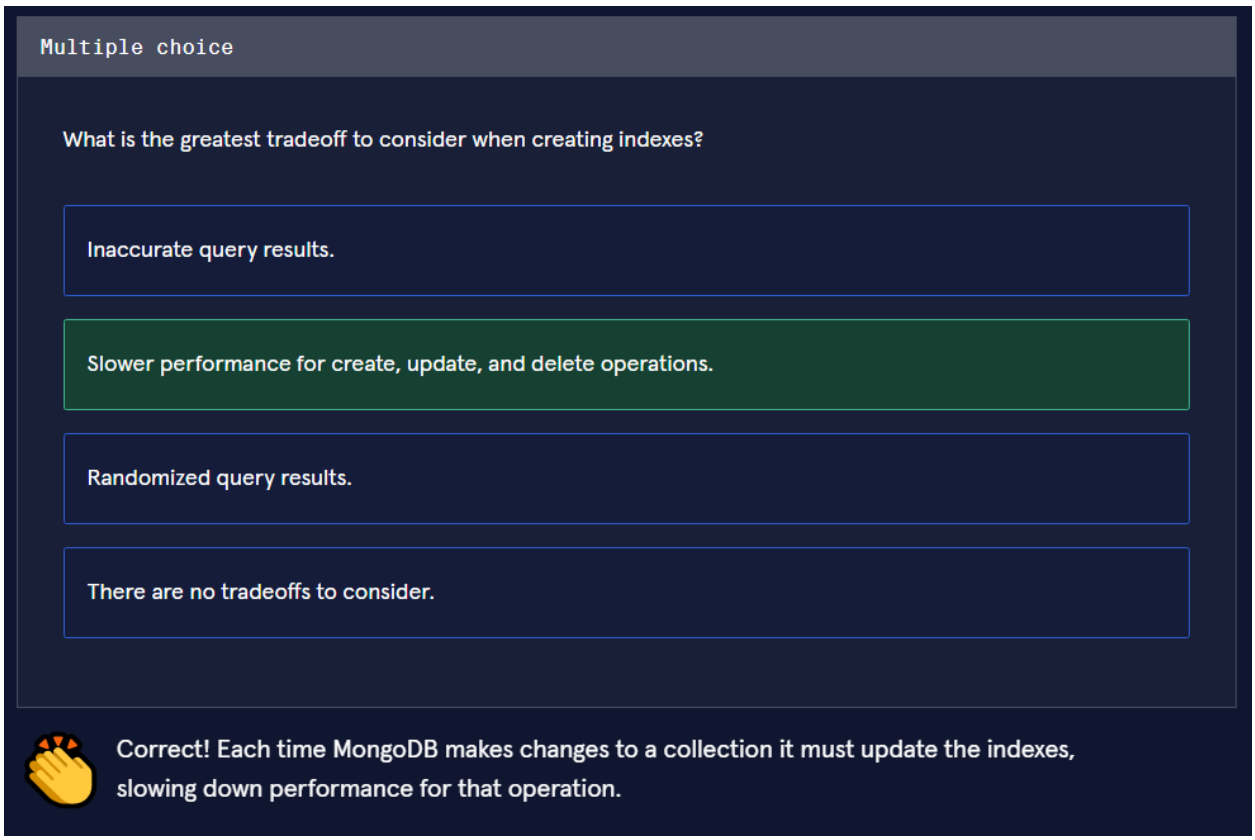
indexes are ordered they can easily sort that subset of matching students chronologically. Notably, the sorted nature of indexes can also improve the efficiency of range-based query operations, where we are seeking to match values that span a given range.

One last type of index worth mentioning is multikey indexes. These indexes support optimized queries on array fields by indexing each element in the array. Conveniently, MongoDB automatically creates a multikey index for us whenever we create an index on a field whose value is an array. Multikey indexes are compatible with both single field and compound indexes.



Tradeoffs and Precautions When Working with Indexes

Indexes are not a cure-all for query optimization and some precautions should be kept in mind. Indexes are most beneficial when they support queries which are selective in nature (the result set represents a small portion of the data in the collection). We should also aim to be conservative, and plan ahead when creating indexes. Each index consumes valuable space. And while indexes can improve query performance, they do so at the cost of write performance. Each time we insert, remove, or update documents in a collection, MongoDB must reflect those changes for each index in the collection, ultimately slowing down the operation. With proper planning, indexes can greatly leverage the power of MongoDB and optimize your queries to improve computing performance, bandwidth, and time efficiency.



Wrap Up

In this article we learned what indexes are, and familiarized ourselves with some of the various types of indexes MongoDB supports. Let's take a moment to recap some of the key takeaways:

- Indexes are data structures that capture a subset of a collection's data in an easy to traverse form.
- Indexes can support more efficient queries of large collections.
- Single field indexes reference one field from a document, while compound indexes reference multiple fields.
- The primary advantage of indexes is that they support faster queries. The greatest tradeoff of indexes is that they can marginally decrease write performance.

Continue to the Introduction to Indexing lesson to gain some hands-on experience with indexes in MongoDB!