VERIOUS Beta Categories □

The Joy of Indexing

Kyle Banker

We spend quite a lot of time at 10gen supporting MongoDB users. The questions we receive are truly legion but, as you might guess, they tend to overlap. We get frequent queries on sharding, replica sets, and the idiosyncrasies of JavaScript, but the one subject that never fails to appear each day on our mailing list is indexing.

Now, to be clear, I'm not talking about how to create an index. That's easy. The trouble runs much deeper. It's knowing how indexes work and having the intuition to create the best indexes for your queries and your data set. Lacking this intuition, your production database will eventually slow to a crawl, you'll upgrade your hardware in vain, and when all else fails, you'll blame both gods and men.

This need not be your fate. You *can* understand indexing! All that's required is the right mental model, and over the course of this series, that's just what I hope to provide.

But *caveat emptor*: what follows is a thought experiment. To get the most out of this post, you can't skim it. Read every word. Use your imagination. Think through the quizzes. Do this, and your indexing struggles may soon be no more.

Picturing an index

To understand indexing, you need a picture in your head. So imagine a cookbook. And not just any cookbook. A massive cookbook, five thousand pages long with the most delicious recipes for every occasion, cuisine, and season with all the good ingredients you might find at home. *This* is the cookbook to end them all. Let's call it The Cookbook Omega.

Now, although this might be best of all possible cookbooks, there are two tiny problems with the Cookbook Omega. The first is that the recipes are in random order. On page 3,475 you have Australian Braised Duck, and on page 2 there's Zacatecan Tacos.

That would be mangeable were it not for the second problem: the Cookbook Omega has no index.

Quiz:
With no index, how do you find the recipe for Rosemary Potatoes in the Cookbook Omega?
Answer:
Your only choice is to scan through every page of the book until you find the recipe. If the recipe is on page 3,973, that's how many pages you have to look through. In the worst case, you have to look at every single page! The solution is to build an index.

The recipe search

There are few ways you can imagine searching for a recipe, but the recipe's name is probably a good place to start. If we create an alphabetical listing at the end of the book of each recipe's name followed by its page number, then we'll have indexed the book by recipe name. A few entries might look like this:

Tibetan Yak SoufflÃ‰
45
Toasted Sesame Dumplings
4,011
Turkey Ã la King
943
As long as we know the name of recipe, or even the first few letters of that name, we can use this index to quickly find that recipe in the book. If that's the only way we expect to search for recipes, then our work is done.

But, of course, this is unrealistic. Because we can also imagine wanting to find recipes based on the ingredients we have in our pantry. Or perhaps we want to search by cuisine. For those cases, we need more indexes.

Quiz:
With just one index in recipe name, how do you find all the Chicken recipes?
Answer:
Lacking the proper indexes, you have to scan the entire book, all 5,000 pages. This is true for any search on ingredients or cuisine.
So, we need to build another index, this time on ingredients. In this index, we have an alphabetical listing of ingredients each pointing to the page number of recipes containing that ingredient. The most basic index on ingredients would thus look like this:

Cashews
3, 20, 42, 88, 103, 1,215…
Cauliflower
2, 47, 88, 89, 90, 275…
Chicken
7, 9, 80, 81, 82, 83, 84…
Currants

Currants
1,001, 1,050, 2,000, 2,133…
Is the index you thought you were going to get? Is it even helpful?

This index is good if all you need is a list of recipes for a given ingredient. But if you have any other information about the recipe that you want to include in your search, you still have some scanning to do. Once you know the page numbers where Cauliflower is referenced, you then need to go to each of those pages to get the name of the recipe and what cuisine it's part of. This is, of course, better than paging through the whole book, but there are still improvements to be made.

Quiz:
You randomly discovered a great chicken recipe in the Cookbook Omega several months ago, but you've forgotten its name. At this point, there are two indexes, one on recipe name and the other on ingredients. Can you think of a way to use these two indexes in combination to find your long lost chicken recipe?
Answer:
I sure hope not. Think about it. If you start with the index on recipe name, but don't remember the name of the recipe, then searching this index is little better than paging through the entire book. If, on the other hand, you start with the index on ingredients, then you'll have a list of page numbers to check, but those page numbers can in no way be plugged in to the index on recipe name. Therefore, you can only use one index in this case, and it turns out that the one on ingredients is the more helpful of the two.
Discussion:
We commonly encounter users who believe that searching on two fields can be facilitated by creating two separate indexes on those fields. This example should give you some intuition about why this just isn't possible.
Happily, there is a solution to the long lost chicken recipe, and its answer lies in the use of compound indexes.

Compound indexes

The two indexes we've created so far are single-key indexes: they both order just one data item from each recipe. We're now going to build out yet another index for the Cookbook Omega, but this time, instead of using just one data item, we'll use two. Indexes that use more than one key like this are called compound indexes.

This compound index uses both ingredients and recipe name, *in that order*. We'll notate the index like this: `ingredient→name`. Here's what part of this index would look like:

Cashews
Cashew Marinade
1,215
Chicken with Cashews
88
Rosemary-Roasted Cashews
103
Cauliflower
Bacon Cauliflower Salad
875
Lemon-baked Cauliflower
89
Spicy Cauliflower Cheese Soup
47
Currants
Creamed Scones with Currants
2,000
Fettuccini with Glazed Duck
2,133
Saffron Rice with Currants
1,050
The value of this index for a human is obvious. You can now search by ingredient and probably very quickly find the recipe you want. For a machine, it's still valuable for this use case, but only if we can provide the first letters of the recipe name, which will keep the database from having to scan every recipe name listed for that ingredient. This compound index would be especially useful if, as with the Cookbook Omega, we had several hundred (or thousand) chicken recipes.

Quiz:
Remember: with compound indexes, order matters. Imagine a compound index on `name→ingredient`. Would this index be interchangeable with the inverse compound index we just explored?
Answer:
Definitely not. With this index, once we have the recipe name, our search is already limited to a single recipe, a single page in our cookbook. So if this index were used on a search for the recipe "Cashew Marinade" and the ingredient "Bananas" then the index could confirm that no such recipe exists. But that's not exactly our use case.
Bonus Quiz:
Our cookbook now has three indexes: one on recipe name, one on ingredients, and one on `ingredient→name`. With the compound index in place, is it possible to eliminate either of the first two indexes we created?
Answer:
Yes! We can safely tear out the single-key index on ingredients. Why? Because a search that just specifies an ingredient can use the index on `ingredient→name`. If we know an ingredient, we can, using that compound index, easily get a list of all page numbers containing said ingredient. Look again at the sample entries for this index to see why this is so.
Exercise:
If we only know the recipe name, why can't we use the index on `ingredient→name` to find that recipe's page number?

Selectivity

We're going to cover one more concept that you can use to design better indexes. It's the idea that some keys for

your data will be more selective than others. Imagine that the recipes in the Cookbook Omega consist of a total of 200 different ingredients but only represent 12 different cuisines. If the cookbook contains 5,000 recipes, which key — ingredients or cuisine — is more selective?

Intuitively, it should be easy to see that ingredient narrows the the number of recipes much more than cuisine does. On averge, there will be 417 (5,000 / 12) recipes per cuisine but only 125 recipes per ingredient (5,000 / 200 * 5). This assumes an average of five ingredients per recipe, but clearly, the actual selectivity of any given ingredient will always be hard to estimate. Some ingredients (chicken) will be less selective than others (anise). But ingredient is generally the more selective of the two fields.

Quiz:
Suppose you want to search the Cookbook Omega on both cuisine and ingredient. You'll need to build a compound index, but the ordering will make a difference. Which should come first? (Hint: think about selectivity).
Answer:
You're probably better off building the compound index on `ingredient→cuisine`. In addition to providing a more selective first key, this compound index will actually be useful for queries on ingredient only. It's easy to see how the inverse compound index on cuisine→ingredient wouldn't be all that useful for the opposite case, where the search is on cuisine alone. (And of course, we all know now that queries on a compound index aren't possible if all we have is the second key.)
Exercise:
Visualize for yourself the difference between the representatons of `ingredient→cuisine` and `cuisine→ingredient` as they would appear as indexes in the Cookbook Omega.

Lessons and Limits

The goal of this post was to lay a groundwork for readers needing a better mental model of indexes. Having a solid mental model is always better than memorizing rules of thumb, but just to help out, here are few concrete lessons that can derived from this thought experiment:

- Indexes make for fast retrieval. Without the proper indexes, you're forced to manually page through your data, which is slow.
- Indexes on a single key cannot be used in combination with one another. For queries that use multiple keys (e.g., ingredient and recipe name), you need a compound index.
- An index on `ingredient` can and should be eliminated if you have a second index on `ingredient→cuisine`. More generally, if you have an index on `a→b`, then a second index on `a` alone will be redundant.
- The order of keys in a compound index matters greatly. It's usually better if the more selective key comes first. (But what's best will always be dicated by the range of queries you expect to perform.)
- Keep a mind to selectivity. You should probably avoid creating indexes that aren't very selective, particularly if they're on a single key only.

One final note is to bear in mind that the cookbook analogy can be taken only so far. It's a model for understanding indexes, but it doesn't fully correspond to the way that B-tree indexes actually work (B-trees are the data structure used to represent indexes in most databases, including MongoDB). The ideas presented here generally hold true, but if you'd like more nuance, stay tuned for the next post, where I'll introduce B-trees and build on the mental model begun here.