

# Indexes

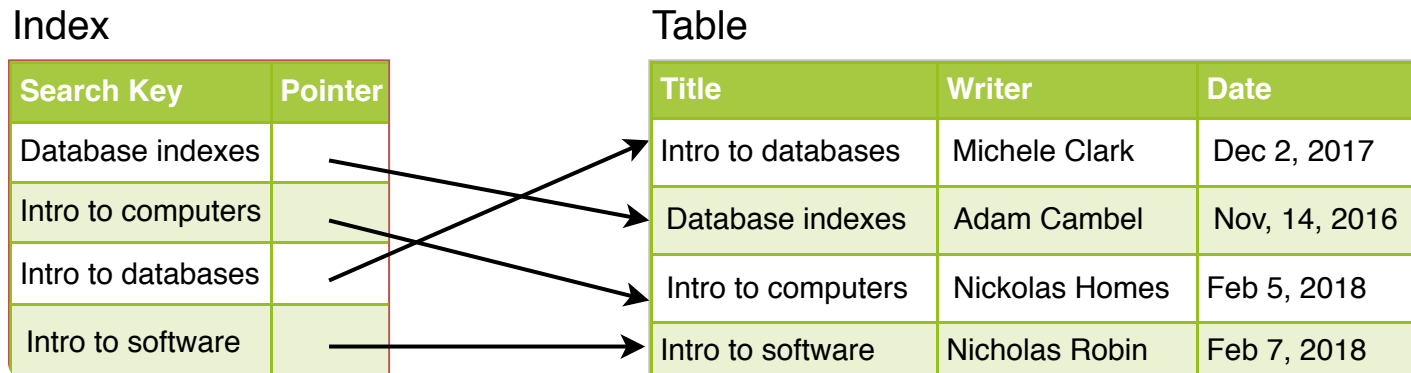
Indexes are well known when it comes to databases. Sooner or later there comes a time when database performance is no longer satisfactory. One of the very first things you should turn to when that happens is database indexing.

The goal of creating an index on a particular table in a database is to make it faster to search through the table and find the row or rows that we want. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

## Example: A library catalog

A library catalog is a register that contains the list of books found in a library. The catalog is organized like a database table generally with four columns: book title, writer, subject and date of publication. There are usually two such catalogs, one sorted by the book title and one sorted by the writer name. That way you can either think of a writer you want to read and then look through their books, or look up a specific book title you know you want to read or in case you don't know the writer's name. These catalogs are like indexes for the database of books. They provide a sorted list of data that is easily searchable by relevant information.

Simply saying, an index is a data structure that can be perceived as a table of contents that points us to the location where actual data lives. So when we create an index on a column of a table, we store that column and a pointer to the whole row in the index. Let's assume a table containing a list of books, following diagram shows how an index on 'Title' column looks like:



Just as to a traditional relational data store, we can also apply this concept to larger datasets. The trick with indexes is that we must carefully consider how users will access the data. In the case of data sets that are many terabytes in size but with very small payloads (e.g., 1 KB), indexes are a necessity for optimizing data access. Finding a small payload in such a large dataset can be a real challenge since we can't possibly iterate over that much data.

For optimizing data access, finding a small payload in such a large dataset can be a real challenge since we can't possibly iterate over that much data

in any reasonable time. Furthermore, it is very likely that such a large data set is spread over several physical devices — this means we need some way to find the correct physical location of the desired data. Indexes are the best way to do this.

## How do Indexes decrease write performance?

The downside is that indexes make it slower to add rows or make updates to existing rows for that table since we not only have to write the data but also have to update the index. So adding indexes can increase the read performance, but at the same time, decrease the write performance.

Adding a new row to a table without indexes is simple. The database finds the next available space in the table to add the new entry and inserts it there. However, when adding a new row to a table with one or more indexes, the database not only have to add the new entry to the table but also have to add a new entry into each index on that table, making sure to insert the entry into the correct spot in the index to ensure that the data is sorted correctly. This performance degradation applies to all insert, update, and delete operations for the table. For this reason, adding unnecessary indexes on tables should be avoided, and indexes that are no longer used should be removed. To reiterate, adding indexes is about improving the performance of search queries. If the goal of the database is to provide a data store that is often written to and rarely read from, in that case, decreasing the performance of the more common operation, which is writing, is probably not worth the increase in performance we get from reading.

For more details, see [Database Indexes](#).

Well you're at the end of this section, did you complete it?

YES!

