Indexes

Indexes are well-known when it comes to databases.

Sooner or later, there comes a time when DB performance is no longer satisfactory. One of the very first thing we should turn to when this occurs 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 catalogue

A library catalogue is a register that contains the list of books found in a library.

The catalogue is organised like a database table generally with four columns: book title, writer, subject, and date of publication.

There are usually two such catalogues: one sorted by the book title, and one sorted by the writer name.

That way, we can either think of a writer we want to read, and then look through their books, or look up a specific book title we want to read if we don't know the writer's name.

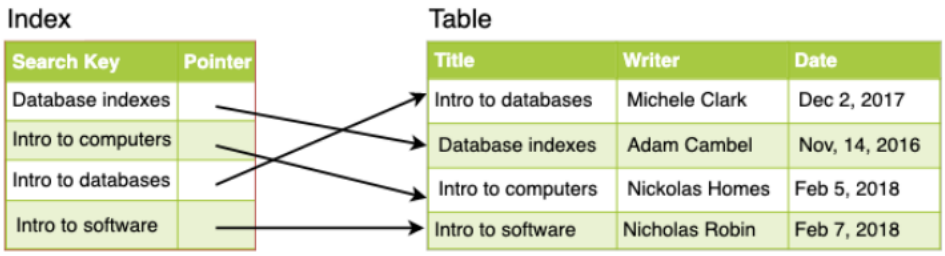
These catalogues are like indexes for the database of books.

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 exists, the following diagram shows what an index on the ‘Title’ column looks like:



Just like 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 datasets that are many terabytes in size, but have very small payloads (e.g. 1KB), indexes are a necessity for optimising 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 dataset is spread over several physical devices — this means that 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?

An index can dramatically speed up data retrieval, but may itself be large due to the additional keys, which slow down data insertion and update.

When adding rows or making updates to existing row for a table with an active index, we not only have to write the data but also have to update the index.

This will decrease the write performance.

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*.

It is typically our goals to improve the performance of the most common operation.

If the goal of the database is to provide a data store that is often written to and rarely read from, adding an index would decrease the performance of the most common operation, so is likely not worth it for the speedups in reading performance.