THOUGHTS AND THEORY

How your data is stored on disk and memory?

Delving a bit into what's happening under the hood when you manipulate your Excel file or data frames.

Guangyuan(Frank) Li Mar 1 · 6 min read

Photo by Christian Wiediger on Unsplash

I consider myself as an applied programmer, I use high-level programming languages like Python to manipulate the data frames, build simple pipelines to facilitate the data analysis of certain tasks. However, at the atomic level, the computer itself doesn't understand what "pandas.read_csv()" means or what the "write.table()" function actually does. It makes me wonder about the intermediate steps involved in converting our commands to 0/1 binary bits and prompts me to write this article.

In this article, I am hoping to give you a rough and intuitive image of what the disk actually looks like, how your data lays out on the disk, and what's the information exchanges between your disk and memory. **This is not meant to be a strictly technical blog but more like relaxing reading material**. Being aware of the physical hardware structure can help us to get a better understanding of the tasks and problems we may encounter in our work:

> *memory leakage? indexing file? memory overhead? I/O streaming?*

So brace up yourself and let's get into that!

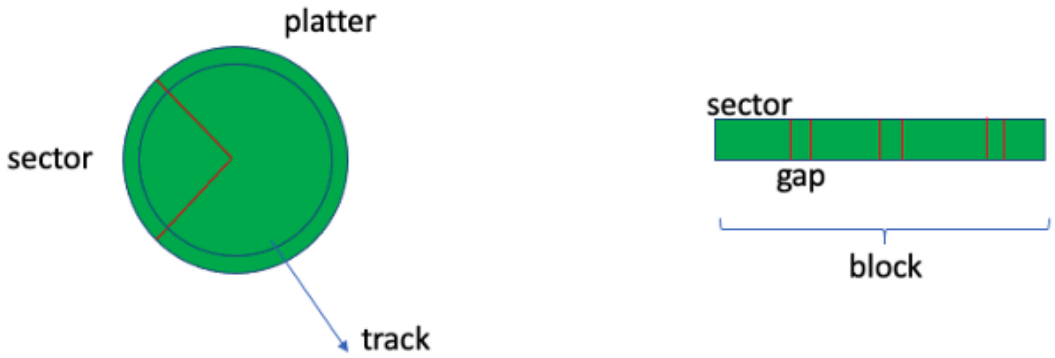
. . .

What the disk looks like?

When you are on your PC or Mac, you open the Computer or Finder, all the files that appear there are stored on the disk, sometime the disk is also referred to as storage. Nowadays, nearly most personal computers are using Solid State Disk (SSD), however, to convey some intuition, I am going to use Hard Disk (HD) to illustrate the physical structure of the disk.

Photo by benjamin lehman on Unsplash

The structure of a typical hard disk looks like a CD player, it has an arm that can rotate and changes to a selected track, the head of the arm is able to read, write or erase the data stored in each track of the surface.



platter, track, sector, gap, block

The surface is also referred to as a platter, and a platter can be split into a set of **physical units** called a sector. There are gaps between each sector and several sectors constitute a block, the block is the **logical unit**.

If I tell you one track in a platter has a maximum capacity of 1024Kb (1Mb), then you know your Excel file, which let's imagine is 2Mb, will occupy two platters to store it in the disk.

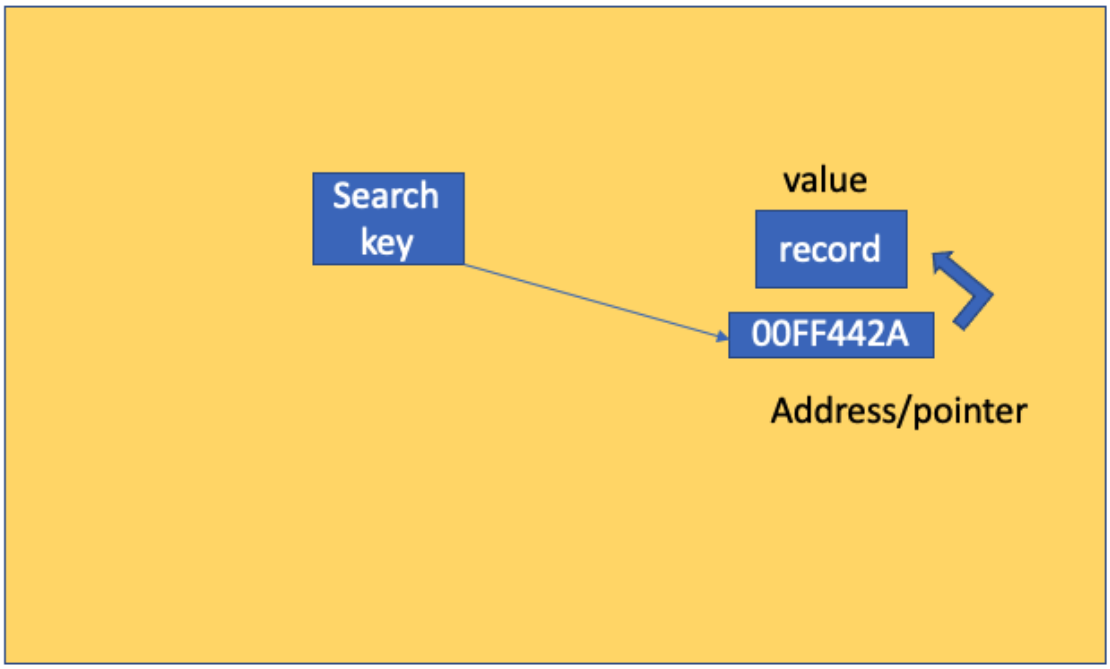
If the rotation speed is known, then you can easily compute how fast the files/bits can be transferred from the disk to memory or vice versus. It also becomes clear that why a

large file needs more time to read because it needs longer time to rotate the arms and arms also take longer to read since the storage area will be bigger.

Solid State Disk (SSD) will have faster accessing time, and the architecture is quite different from the hard disk (HD) example that I showed you above. But hopefully, that can provide you a clear image that **where** your files lie on, and next I will show you **how** the data actually lies on the disk.

How the data lays out on the disk?

In a Database Management System (DBMS), each record (a record is basically a row in a data frame) lies on disk in some particular manner to speed up the search, insert, and deletion. Imagine if every record is scattered around the disk without any rules or organizations, there will be no way to perform the query, indexing, and any manipulations.



A single record/data on disk

Simply speaking, you can store all the data in a row-wise manner, meaning to say that similar records/rows will be gathered together. In doing so, row-wise indexing can be very efficient with the cost of adding column operation or column-wise indexing can be

relatively slow. If you go along the column-wise manner, things would become exactly the opposite.

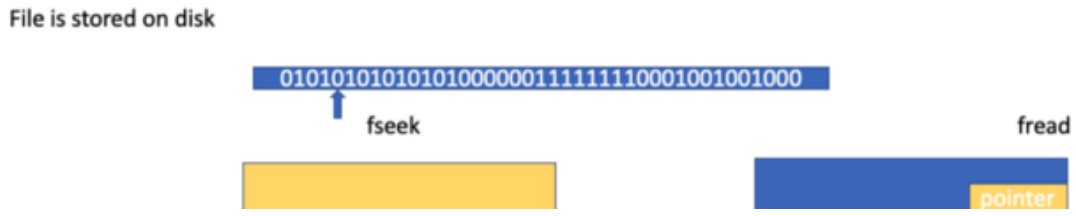
Let's imagine each row will be stored in a coalesced space on disk, as shown in the figure. The physical address or pointer is associated with this record. In addition to that, a search key is usually present to serve as a proxy in the **index file**. An index file is basically pairs of search key and pointer combinations. So having a search key aims to expedite the searching and avoid the overhead. With the presence of the search keys, scientists have developed a wide array of efficient schemas to represent and organize all the search keys. Prominent examples include:

1. **B+ tree**: self-balanced tree such that the distance from the root to all leaf nodes is equal.
2. **B tree**: Similar to B+ tree but it has record stored on non-leaf nodes
3. **Extensible hashing**: a directory together with buckets, buckets store the hashing key computed by hash table or hash function.
4. **Linear hashing**: Similar to extensible hashing without directory but multiple levels of hash functions.

Since it is not a technical blog, I would like to refer you to some wonderful youtube videos (link above) that explain what each indexing schema is and how they enable quick and efficient insertion and deletion. But the take-home message from this section is, **data is laid out on disk in a well-designed manner that enables us to efficiently fetch and manipulates them.**

How to read bytes/files from disk to memory?

First, I am hoping to instill a little bit of non-intuitive fact that **a file is actually a stream of bits/bytes**. When you read a file into memory, you are dealing with a series of 0/1s and you are hoping to load a specific portion of files correctly.



DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.