When you add an index to a collection it searches the collection for all relevant fields within all the documents within a collection and saves those fields and associated values in a format that is easily traversable (Binary-tree format). Each node (the node is the “box” that the field and value are stored) contains a pointer to the document as a whole. This means that each pointer, and by extension each document, is stored and organized within the binary-tree based on a specific field so that retrievals can be done on that specific field in a more efficient manner.

Index cardinality is defined as the number of unique field values within a collection. For example, if you index a Boolean field the index cardinality is two because there are only two possible distinct values. Index cardinality is important because indexing a field such as a Boolean field is not going to improve performance as much as indexing a field such as a string field. This is because of the way a binary-tree organizes its nodes and optimizes searching for each node (Click link below on efficiency of binary-search-tree to understand what I mean.) If there are only two distinct values by which to organize the fields within a binary-tree, there is no way to optimize searches.

Capped collections are collections that are capped or can only hold a certain number of documents before the oldest one is deleted. Because each document is stored in the order of creation, writing documents becomes much more efficient (high insertion throughput) than if indexes were to be used (I believe this is due to the fact that indexes require the program to search for the correct place to put a node within the binary-tree which is less efficient than simply adding it to the front of a list of nodes, couldn’t find any helpful video or article explaining the process so this is just an assumption). This type of collection is useful for storing logs and cached data which may require many, many writes, and benefit from the auto-delete feature.

A sparse index saves only the pointers of the documents that actually contain a given field within the binary-tree. A traditional index saves pointers of all the documents regardless of whether or not they contain a given field. A traditional index stores null values for documents that do not contain the field in question.

Efficiency of Binary-search-tree (not binary-tree but similar concept) explained:

<https://duckduckgo.com/?q=binary+tree+efficiency&atb=v314-1&iar=videos&iax=videos&ia=videos&iai=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3Duvxmgr5Doao>

B-tree and MongoDB (start from 11:05 if you are only interested in how a b-tree works):

<https://duckduckgo.com/?q=indexing+mongodb&atb=v314-1&iax=videos&iai=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DIHQeDEn38BQ&ia=videos>

More on indexing:

<https://duckduckgo.com/?q=indexing+mongodb+btree&atb=v314-1&iar=videos&iax=videos&ia=videos&iai=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DtSgPhxZdhLk>

Index cardinality:  
<https://bharatkalluri.com/posts/cardinality-and-indexing-mongodb>

Capped index:

<https://www.mongodb.com/docs/manual/core/capped-collections/>

Sparse vs traditional indexing:

<https://www.mongodb.com/docs/manual/core/capped-collections/>