The idea behind an index is very intuitive: think at a book and its appendix at the end; the appendix gives you a list of keywords, ordered alphabetically, and the pages where those words are referenced in the book. If we now translate this to DBMS, the index is just a tool to search for a certain term and find some addresses in a file(I, therefore, mean physical addresses on the disk) corresponding to records that have that term incorporated.

The alternative, if no indexing is done, is just to look linearly through the file and pick the records that satisfy your criterion.

An index typically stores two things: the value of the index field and a list of pointers to all disk blocks that contain records corresponding to the field specified. Usually the index field is sorted, so a binary search could be performed on it (rather than doing a linear one).

Having this in mind, let's move forward. We have 3 types of indexes:

1) PRIMARY index, which is specified on the ordering key of an ordered file (remember that an ordering key is the field used to physically order the records in a file on the disk)

2) CLUSTERING index, which is specified on an ordering field which is not a key. Since the fields that are not keys can have repeated values(e.g.: a lot of people with the name "Mark") we can view this as a cluster of records with the same index field value.

3) SECONDARY index, which can be specified on any non ordering field.

Let's dive a little bit more into details:

----------PRIMARY INDEX----------

This kind of index is a data structure(or a file) with only two fields: the first field is of the same type as the ordering key of the file we are indexing(thus, the primary key) and the second field is a pointer to a disk block.

OK. But what disk block?

As stated, a primary index has two fields, BUT N records, where N is the number of disk blocks occupied by the file. So a primary index has the number of rows equal to the number of disk blocks of the file we are indexing. Thus, the second field of a primary index is an address to the starting point of each disk block of the file. But what about the first field? The first field is the value of the ordering key of the first record that appears in that block. (also called the anchor record)

We now have the following form:

---> index[i] = {value of the primary key of the first record of the i-th block; address of the i-th block}

One can now easily see where this goes: when we search for a record, we do a binary search of the index comparing the primary key of the record we are searching for with the index's stored ordering keys. Once we found a match, we return the disk block's address and we can do a second binary search there(since the file is ordered).

This type of index is sparse, since we have as many entries in it as disk blocks, while a disk block contains several records. The primary index occupies a lot less space than the actual file so searching is much(very much) faster.

Disadvantages? Well, the index is an ordered file itself and as it is with any ordered file, insertion and deletion are a problem. Why? Well, if we insert or delete a record in the file we are indexing, we might reposition some records in the disk blocks, thus changing the anchor records, so we need to update the primary index as well.

In conclusion, a primary index makes data retrieval extremely fast. (O(log(B)), where B is the number of disk blocks of the data we are indexing, but it is pretty inefficient when we insert or delete records in the file since we have to remap the anchor records.

----------CLUSTERING INDEX----------

This is used if the file is ordered by a non-key field. (any ordering field but a key(including superkeys) of the file) This field will be known as clustering field, because, as I have mentioned above, many records in the file can have the same value for the clustering field. (some ML helps here)

What is the structure of this index? Again, it is an ordered file with two fields: the first being of the same data type as the one of the clustering field and the second one being a pointer to a disk block.

What disk block? The disk block that contains the FIRST apparition of a record for which the clustering field agrees(in value) with our search. We have as many entries as there are values in the domain of the clustering field.

Thus, the following form:

---> index[i] = {the i-th value from the domain of the clustering index; the address of the first disk block that contains a record that matches the search}

This is also an example of non dense index, since we do not have an index entry for each data record, but for each possible and distinct value of the clustering index.

Disadvantages? Note that deletion and insertion still pose a problem. This type of index is very similar to hashing. I can arguably say that this kind of index would be even faster than a primary index, if the clustering field is of some type that can easily cope with a hashing function, thus making retrieval of data in O(1).

----------SECONDARY INDEX----------

A secondary index provides a secondary means for accessing a data file for which a primary index already exists. (recall that each file can have a SINGLE primary index, a SINGLE clustering index and MORE secondary indexes).

You can create a secondary index of any non ordering field. If the field is unique for every record, you will have as many entries in the secondary index as there are in the original file. We call this a dense index.

A secondary index usually needs more storage space and longer search time. Not that this index is still an ordered file, so insertion and deletion will still have to suffer. So why would we use this?

Main reason is that you can create a secondary index on any kind of data: heap, hashed or sequential. (primary and clustering indexes can only be created on sequential files).

Also, not that the secondary index provide a logical ordering of a file after a certain indexing field, while the file is ordered probably after the ordering key (in the case of ordered files). By creating several secondary indexes, we can provide many more logical orderings, thus enabling faster retrieval time on unordered files.

-------------------------------------