Indexing

Indexing in simple terms is to reorganise/process data items in a database such that we have a key to data in the db that helps us to find the location of the item in the memory. In indexing we take a search query as input and efficiently return the collection of matching records.
We calculate this index(key) by several techniques such that the searching process is as time efficient and space efficient as possible.

We have different types of indexing in practise

Dense Index : All the data is sorted based on an attribute that is unique, let's say primary key. And then we have an index record(that has the key and holds the pointer to the data with the key) for every search key in the data file .This helps us search faster but needs more space to store index records.
Sparse Index : In this we store only some index records pointing to the start of blocks which are like containers of data. This is to have better space complexity than dense indexing but the search time is slower.So we dont reference all keys in this indexing .Pointers point only to the first key of the block .So we can't even say from the index if a key is present or not,until we follow the pointer and check through the block
Secondary Indexing : This is kind of a two level indexing technique and is used to reduce the mapping size of first level . Between index records and actual data blocks we have buckets in the middle .We then have indexes on these buckets and these buckets contain pointers to the actual data.It is kind of a mix of sparse(index on buckets) and dense(buckets to data) indexing to be more efficient.
Hashing : In this we have a Hash Function that maps a key to a bucket in which we store actual data . Simply when we get a query key the output we get after putting it in the hash function gives us the container where we should look for .

B-Tree(B+ Tree) : It is a data structure for tree based indexing .It is constructed in such a way that all leaf nodes contain actual data pointers with key values in them .And all leaf nodes are connected like a linked list to support random and sequential access .And the non-leaf nodes contain keys and pointers to guide us navigate to the required value .If a non-leaf node has n search keys it has n+1 pointers(a pointer to left and right of every key). If the query key is less than the search key we go to the left pointer of it otherwise right.

kd-Trees : It is a generalisation of binary search trees to multidimensional data .It is a space partitioning data structure in a k-dimensional space .A non-leaf node in kd-tree divides the space into two parts . Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree . Each level in the tree goes on dividing the plane wrt to separate planes and each non-leaf node contains the plane/attribute on which we divided the space, a dividing value and two left and right pointers . And leaves are the blocks with records in the space we arrived at.