File Organization

A database consist of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records. A user can see that the data is stored in form of tables, but in acutal this huge amount of data is stored in physical memory in form of files.

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.

What is File Organization?

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in certain order is called file Organization. **File Structure** refers to the format of the label and data blocks and of any logical control record.

Types of File Organizations –

Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection . Thus it is all upon the programmer to decide the best suited file Organization method according to his requirements.

Some types of File Organizations are :

- Sequential File Organization
- Heap File Organization
- Hash File Organization

We will be discussing each of the file Organizations in further sets of this article along with differences and advantages/ disadvantages of each file Organization methods.

Sequential File Organization –

The easiest method for file Organization is Sequential method. In this method the the file are stored one after another in a sequential manner. There are two ways to implement this method:

1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.

   Insertion of new record –
   Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

2. **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records

may be based on any primary key or any other key.

Insertion of new record –

Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .

Pros and Cons of Sequential File Organization –

Pros –

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Cons –

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

Heap File Organization –

Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.

Insertion of new record –

Suppose we have four records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the database selected by the DBMS, lets say data block 1.

If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Pros and Cons of Heap File Organization –

Pros –

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Cons –

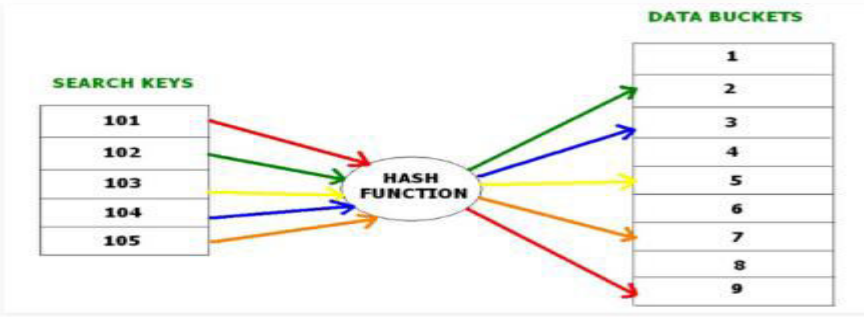
- Problem of unused memory blocks.
- Inefficient for larger databases.

In database management system, When we want to retrieve a particular data, It becomes very inefficient to search all the index values and reach the desired data. In this situation, Hashing technique comes into picture.
Hashing is an efficient technique to directly search the location of desired data on the disk without using index structure. Data is stored at the data blocks whose address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket.

Hash File Organization :

- **Data bucket –** Data buckets are the memory locations where the records are stored. These buckets are also considered as *Unit Of Storage*.
- **Hash Function –** Hash function is a mapping function that maps all the set of search keys to actual record address. Generally, hash function uses primary key to generate the hash index – address of the data block. Hash function can be simple mathematical function to any complex mathematical function.
- **Hash Index-**The prefix of an entire hash value is taken as a hash index. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed ? then the depth value is increased linearly and twice the buckets are allocated.

Below given diagram clearly depicts how hash function work:



Hashing is further divided into two sub categories :

Static Hashing –

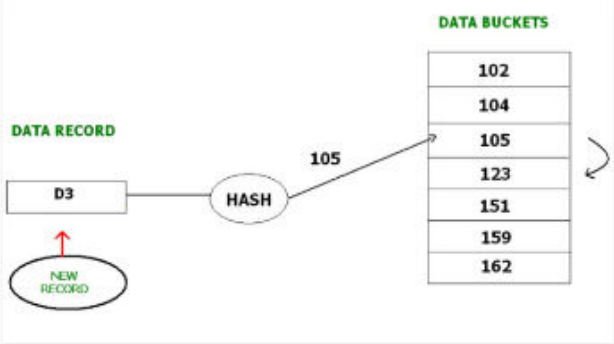
In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if we want to generate address for STUDENT_ID = 76 using mod (5) hash function, it always result in the same bucket address 4. There will not be any changes to the bucket address here. Hence number of data buckets in the memory for this static hashing remains constant throughout.

Operations –

- **Insertion –** When a new record is inserted into the table, The hash function h generate a bucket address for the new record based on its hash key K.
  Bucket address = h(K)
- **Searching –** When a record needs to be searched, The same hash function is used to retrieve the bucket address for the record. For Example, if we want to retrieve whole record for ID 76, and if the hash function is mod (5) on that ID, the bucket address generated would be 4. Then we will directly got to address 4 and retrieve the whole record for ID 104. Here ID acts as a hash key.
- **Deletion –** If we want to delete a record, Using the hash function we will first fetch the record which is supposed to be deleted. Then we will remove the records for that address in memory.
- **Updation –** The data record that needs to be updated is first searched using hash function, and then the data record is updated.

Now, If we want to insert some new records into the file But the data bucket address generated by the hash function is not empty or the data already exists in that address. This becomes a critical situation to handle. This situation in the static hashing is called **bucket overflow**.
How will we insert data in this case?
There are several methods provided to overcome this situation. Some commonly used methods are discussed below:

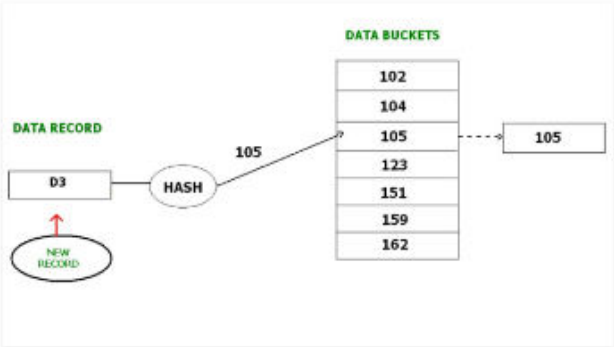
1. **Open Hashing –**
   In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one. This method is also called linear probing. For example, D3 is a new record which needs to be inserted , the hash function generates address as 105. But it is already full. So the system searches next available data bucket, 123 and assigns D3 to it.



2. **Closed hashing –**

In Closed hashing method, a new data bucket is allocated with same address and is linked it after the full data bucket. This method is also known as overflow chaining.

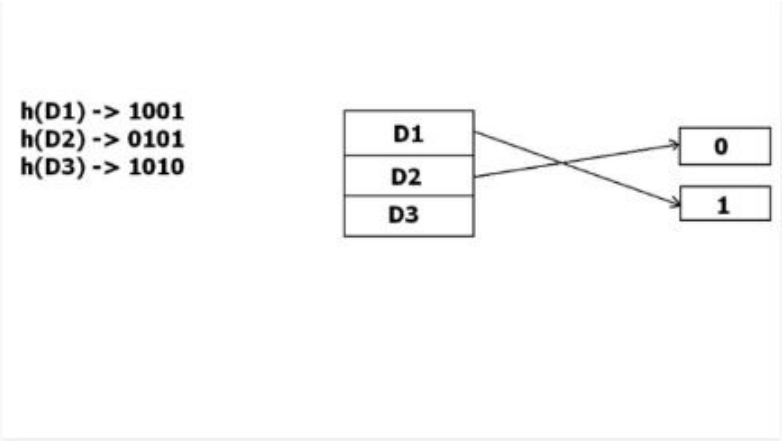
For example, we have to insert a new record D3 into the tables. The static hash function generates the data bucket address as 105. But this bucket is full to store the new data. In this case is a new data bucket is added at the end of 105 data bucket and is linked to it. Then new record D3 is inserted into the new bucket.



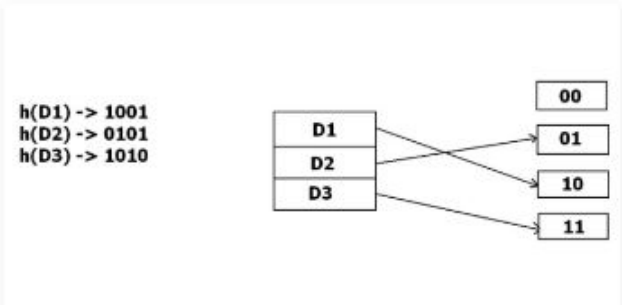
Dynamic Hashing –

The drawback of static hashing is that that it does not expand or shrink dynamically as the size of the database grows or shrinks. In Dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases. Dynamic hashing is also known as extended hashing.

In dynamic hashing, the hash function is made to produce a large number of values. For Example, there are three data records D1, D2 and D3 . The hash function generates three addresses 1001, 0101 and 1010 respectively. This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at address 0 and 1.



But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address. Then it tries to accommodate D3.



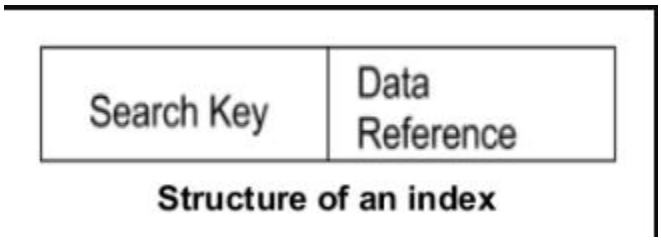
Indexing in Databases

Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed.

An index or database index is a data structure which is used to quickly locate and access the data in a database table.

Indexes are created using some database columns.

- The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly (Note that the data may or may not be stored in sorted order).
- The second column is the Data Reference which contains a set of pointers holding the address of the disk block where that particular key value can be found.



Structure of an index

There are two kinds of indices:

1. **Ordered indices:** Indices are based on a sorted ordering of the values.
2. **Hash indices:** Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by function called a hash function.

Indexing Methods

Ordered Indices

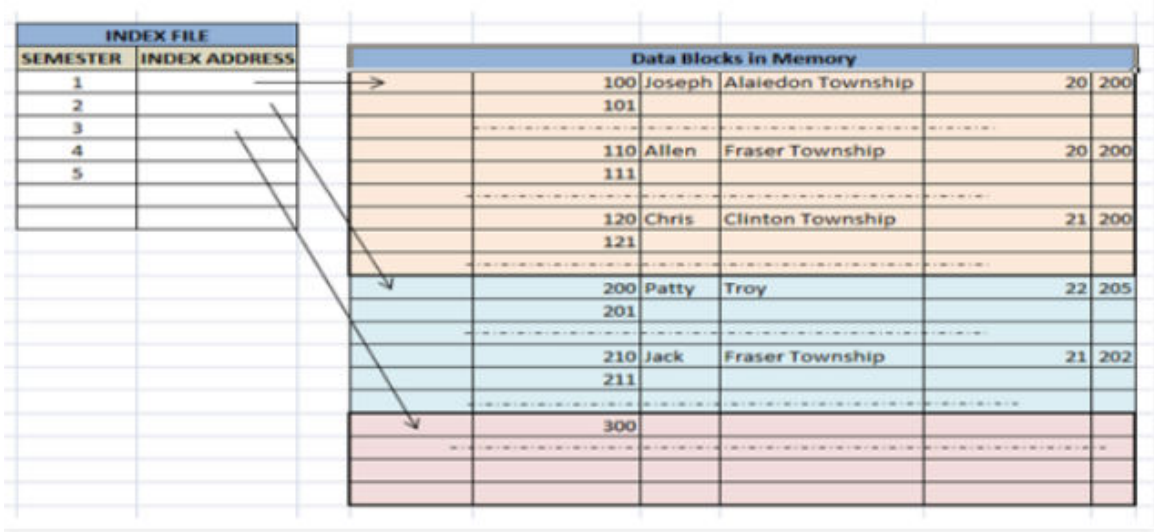
The indices are usually sorted so that the searching is faster. The indices which are sorted are known as ordered indices.

- If the search key of any index specifies same order as the sequential order of the file, it is known as primary index or clustering index.
  Note: The search key of a primary index is usually the primary key, but it is not necessarily so.
- If the search key of any index specifies an order different from the sequential order of the file, it is called the secondary index or non-clustering index.

Clustered Indexing

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.



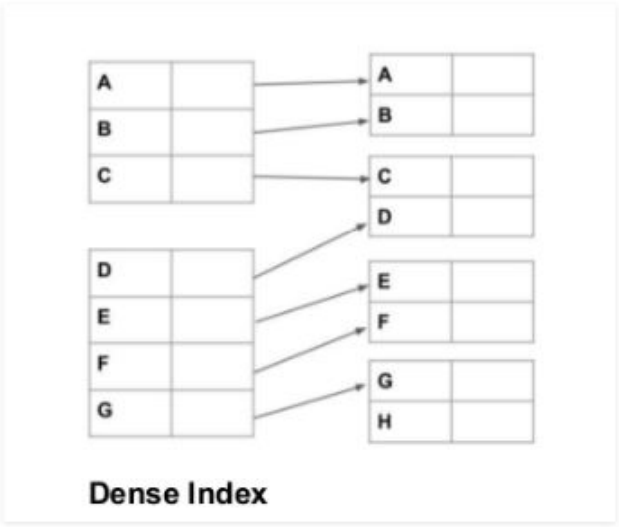
?Clustered index sorted according to first name (Search key)

Primary Index? ?

In this case, the data is sorted according to the search key. It induces sequential file organisation.
In this case, the primary key of the database table is used to create the index. As primary keys are unique and are stored in sorted manner, the performance of searching operation is quite efficient. The primary index is classified into two types : **Dense Index** and **Sparse Index**.

(I) Dense Index : ?

- For every search key value in the data file, there is an index record.
- This record contains the search key and also a reference to the first data record with that search key value.



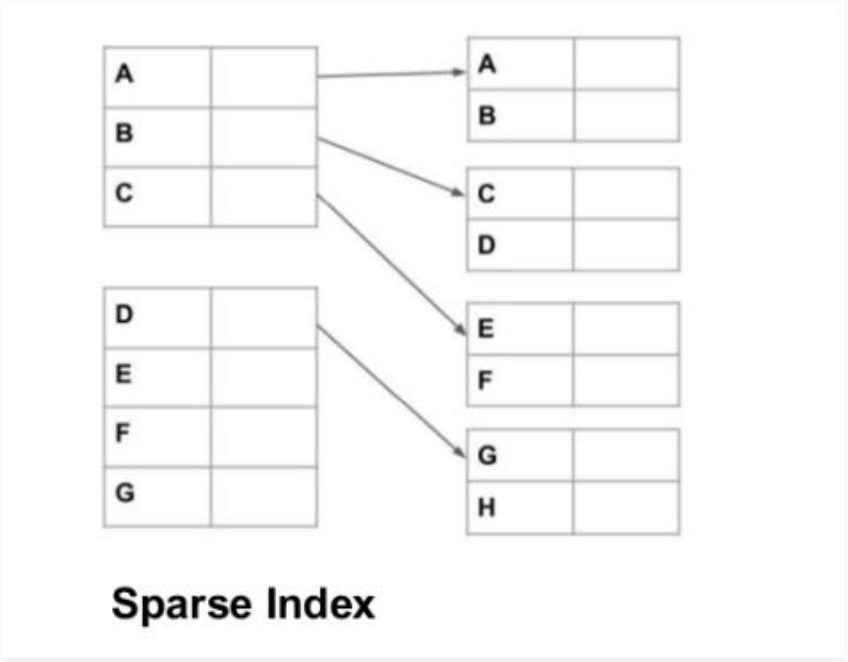
Dense Index

(II) Sparse Index :

?

- The index record appears only for a few items in the data file. Each item points to a block as shown.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is,

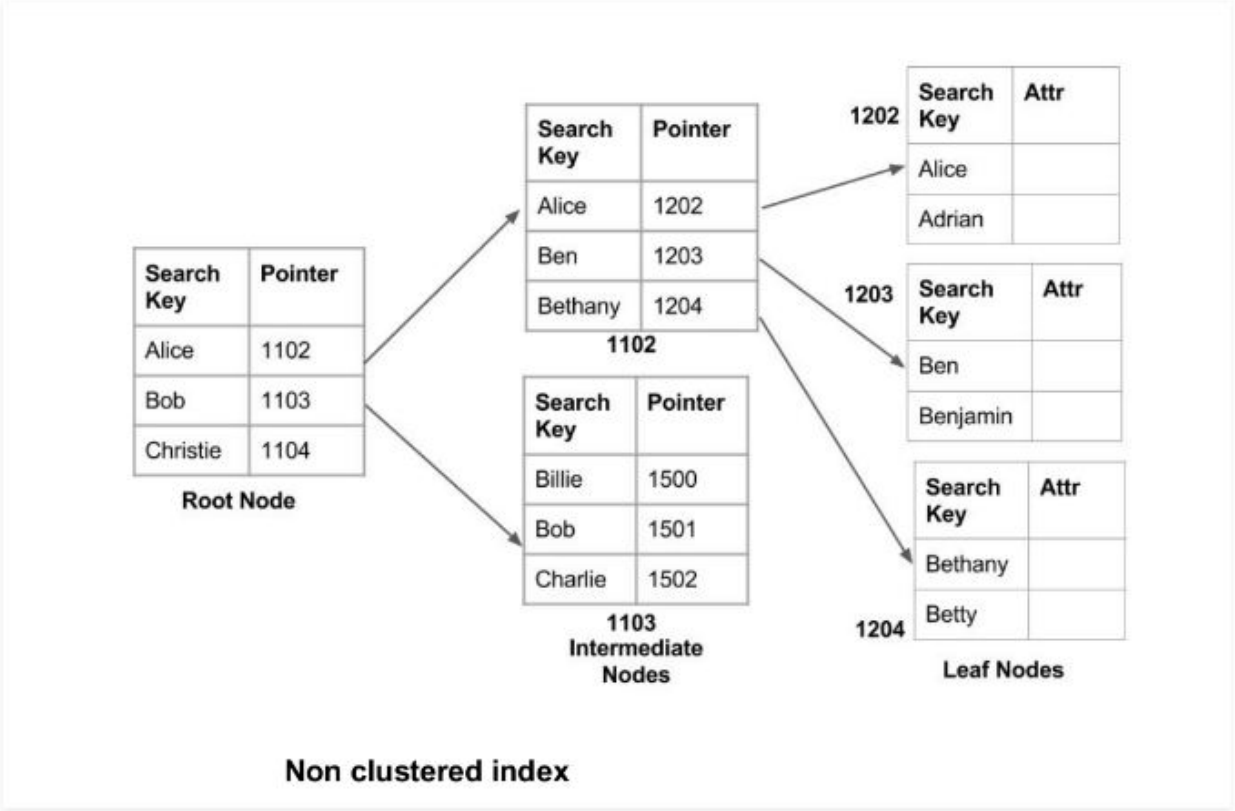
sequentially) until we find the desired record.



Non-Clustered Indexing

?A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead , data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of book) is not organised but we have an

ordered reference(contents page) to where the data points actually lie.

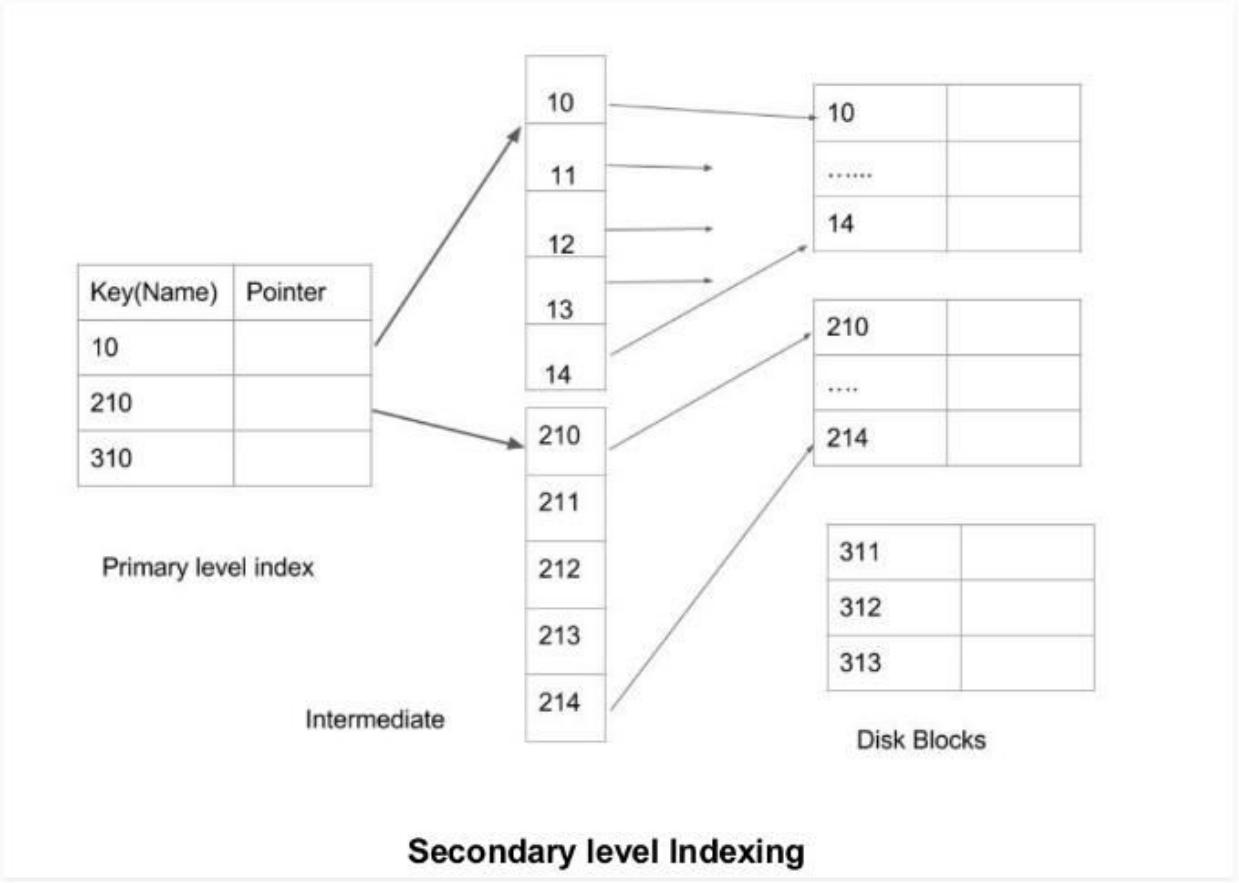


It requires more time as compared to clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In case of clustered index, data is directly present in front of the index.

Secondary Index?

It is used to optimize query processing and access records in a database with some information other than the usual search key (primary key). In this two levels of indexing are used in order to reduce the mapping size of the first level and in general. Initially, for the first level, a large range of numbers is selected so that the mapping size is small. Further, each range is divided into further sub ranges.

In order for quick memory access, first level is stored in the primary memory. Actual physical location of the data is determined by the second mapping level.



Secondary level Indexing