Overview of Physical Storage Media

1. Several types of data storage exist in most computer systems. They vary in speed of access, cost per unit of data, and reliability.
    - **Cache:** most costly and fastest form of storage. Usually very small, and managed by the operating system.
    - **Main Memory (MM):** the storage area for data available to be operated on.
        - General-purpose machine instructions operate on main memory.
        - Contents of main memory are usually lost in a power failure or ``crash".
        - Usually too small (even with megabytes) and too expensive to store the entire database.
    - **Flash memory:** EEPROM (*electrically erasable programmable read-only memory*).
        - Data in flash memory survive from power failure.
        - Reading data from flash memory takes about 10 nano-secs (roughly as fast as from main memory), and writing data into flash memory is more complicated: write-once takes about 4-10 microsecs.
        - To overwrite what has been written, one has to first erase the entire bank of the memory. It may support only a limited number of erase cycles (to).
        - It has found its popularity as a replacement for disks for storing small volumes of data (5-10 megabytes).
    - **Magnetic-disk storage:** primary medium for long-term storage.
        - Typically the entire database is stored on disk.
        - Data must be moved from disk to main memory in order for the data to be operated on.
        - After operations are performed, data must be copied back to disk if any changes were made.
        - Disk storage is called **direct access** storage as it is possible to read data on the disk in any order (unlike sequential access).
        - Disk storage usually survives power failures and system crashes.
    - **Optical storage:** CD-ROM (compact-disk read-only memory), WORM (*write-once read-many*) disk (for archival storage of data), and *Juke box* (containing a few drives and numerous disks loaded on demand).
    - **Tape Storage:** used primarily for backup and archival data.
        - Cheaper, but much slower access, since tape must be read sequentially from the beginning.
        - Used as protection from disk failures!

2. The storage device hierarchy is presented in Figure 10.1, where the higher levels are expensive (cost per bit), fast (access time), but the capacity is smaller.

Figure 10.1: Storage-device hierarchy

3. Another classification: Primary, secondary, and tertiary storage.
    1. Primary storage: the fastest storage media, such as cash and main memory.
    2. Secondary (or on-line) storage: the next level of the hierarchy, e.g., magnetic disks.
    3. Tertiary (or off-line) storage: magnetic tapes and optical disk juke boxes.
4. Volatility of storage. *Volatile storage* loses its contents when the power is removed. Without power backup, data in the volatile storage (the part of the hierarchy from main memory up) must be written to nonvolatile storage for safekeeping.

RAID

Redundant Array of Independent Disk (RAID) combines multiple small, inexpensive disk drives into an array of disk drives which yields performance more than that of a Single Large Expensive Drive (SLED). RAID is also called Redundant Array of Inexpensive Disks.

Storing the same data in different disk increases the fault-tolerance.

The array of Mean Time Between Failure (MTBF) = MTBF of an individual drive, which is divided by the number of drives in the array. Because of this reason, the MTBF of an array of drives are too low for many application requirements.

Types of RAID

The various types of RAID are explained below −

RAID-0

RAID Level-0 is not redundant. Since no redundant information is stored, performance is very good, but the failure of any disk in the array results in data loss. A single record is divided into strips typically 512 bytes and is stored across all disks. The record can be accessed quickly by reading all disks at the same time, called as striping.

RAID-1

RAID Level-1 provides redundancy by writing all data into two or more drives. The performance is faster on reads and slower on writes compared to a single drive. If anyone drives fails, no data is lost. This method is called mirroring.

RAID-2

RAID Level-2 is used for Hamming error correction codes and is used with drives which do not have built-in error detection.

RAID-3

RAID Level-3 stripes data at a byte level across several drives, with parity stored on one drive. Byte-level stripping hardware supports efficient use.

RAID-4

RAID Level-4 that stripes data at a block level across several drives, with parity stored on one drive. Parity information allows recovery from the failure of any single drive. The performance of the level-4 array is good for reads.

Writes, however, require that parity data be updated each time. Because only one drive in the array stores redundant data. The cost per megabyte is low.

RAID-5

RAID Level-5 is similar to level 4, but distributes parity among the drives. This can speed up small writes in the multiprocessing system. The performance for reads is lower than a level-4 array. The cost per megabyte is the same as level-4.

Summary

Given below is the summary of all the types of RAID −

Levels	Summary
RAID-0	It is the fastest and most efficient array type but offers no fault-tolerance.

Levels	Summary
RAID-1	It is the array of choice for a critical, fault tolerant environment.
RAID-2	It is used today because ECC is embedded in almost all modern disk drives.
RAID-3	It is used in single environments which access long sequential records to speed up data transfer.
RAID-4	It offers no advantages over RAID-5 and does not support multiple simultaneous write operations.
RAID-5	It is the best choice in a multi-user environment. However, at least three drives are required for the RAID-5 array.

File Organization in DBMS | Set 1

- Difficulty Level : Easy
- Last Updated : 20 Jul, 2022

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 actual 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 tapes 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
- B+ Tree File Organization
- Clustered 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 file are stored one after another in a sequential manner. There are two ways to implement this method:

- **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.

1. **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.

- **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.

1. **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 data blocks 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 databa

Organization of Records in Files

There are several ways of organizing records in files.

- **heap file organization**. Any record can be placed anywhere in the file where there is space for the record. There is no ordering of records.
- **sequential file organization**. Records are stored in sequential order, based on the value of the search key of each record.
- **hashing file organization**. A hash function is computed on some attribute of each record. The result of the function specifies in which block of the file the record should be placed -- to be discussed in chapter 11 since it is closely related to the indexing structure.
- **clustering file organization**. Records of several different relations can be stored in the same file. Related records of the different relations are stored on the same block so that one I/O operation fetches related records from all the relations.

For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data. Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

Hashing uses hash functions with search keys as parameters to generate the address of a data record.

Hash Organization

- **Bucket** − A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.

- **Hash Function** − A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Static Hashing

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.

Operation

- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.
  Bucket address = h(K)
- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- **Delete** − This is simply a search followed by a deletion operation.

Bucket Overflow

The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.

- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.

Dynamic Hashing

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.

Organization

The prefix of an entire hash value is taken as a hash index. Only a portion of the hash value is used for computing bucket addresses. 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 − that is, when all the buckets are full − then the depth value is increased linearly and twice the buckets are allocated.

Operation

- **Querying** − Look at the depth value of the hash index and use those bits to compute the bucket address.
- **Update** − Perform a query as above and update the data.
- **Deletion** − Perform a query to locate the desired data and delete the same.
- **Insertion** − Compute the address of the bucket
    - If the bucket is already full.
        - Add more buckets.
        - Add additional bits to the hash value.
        - Re-compute the hash function.
    - Else
        - Add data to the bucket,
    - If all the buckets are full, perform the remedies of static hashing.

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexit

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

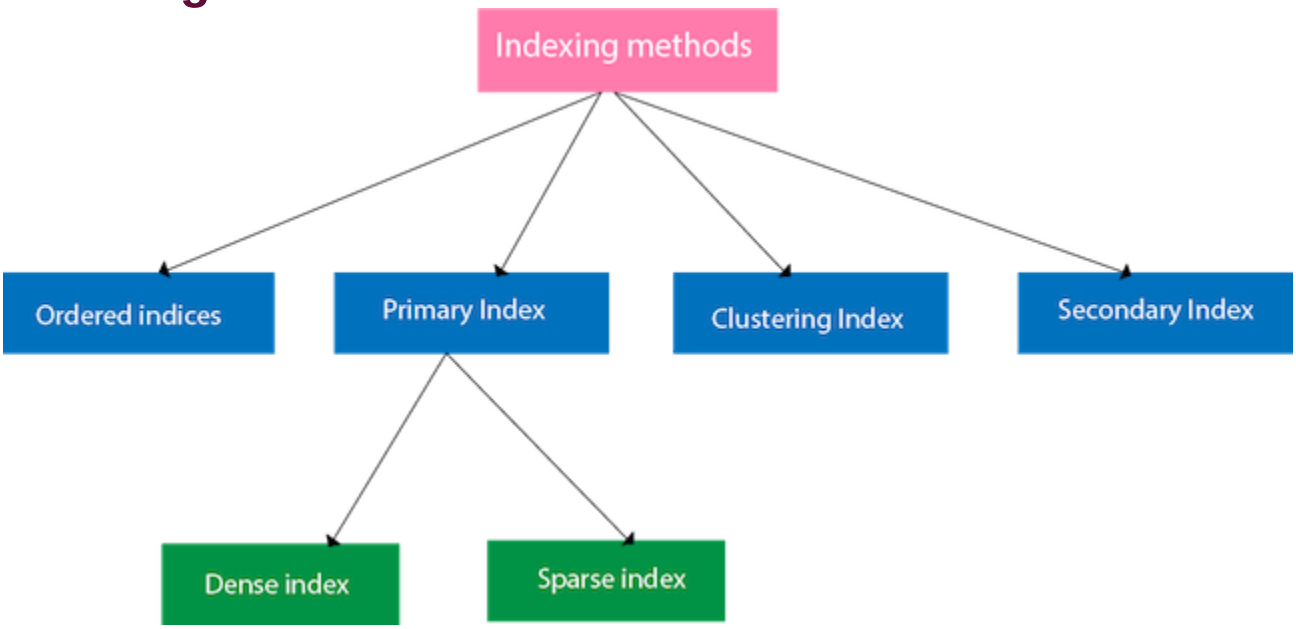
Indexes can be created using some database columns.

Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

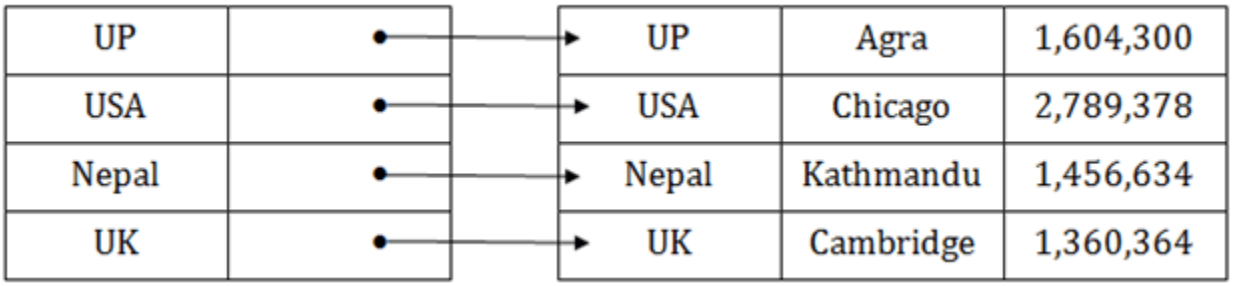
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

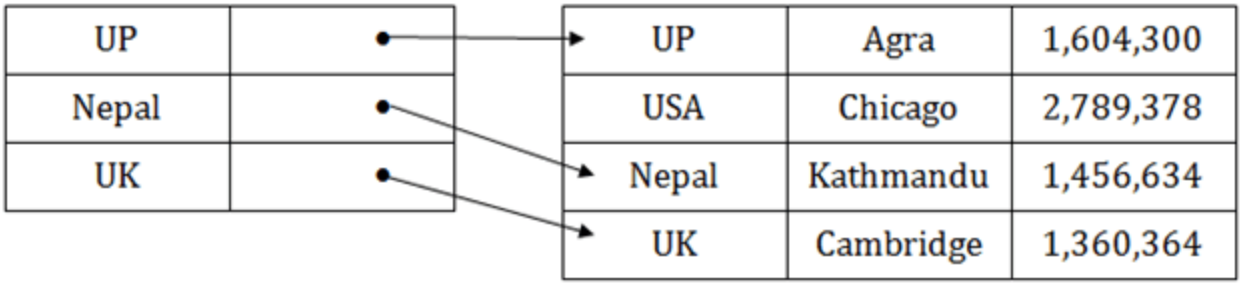
Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



Sparse index

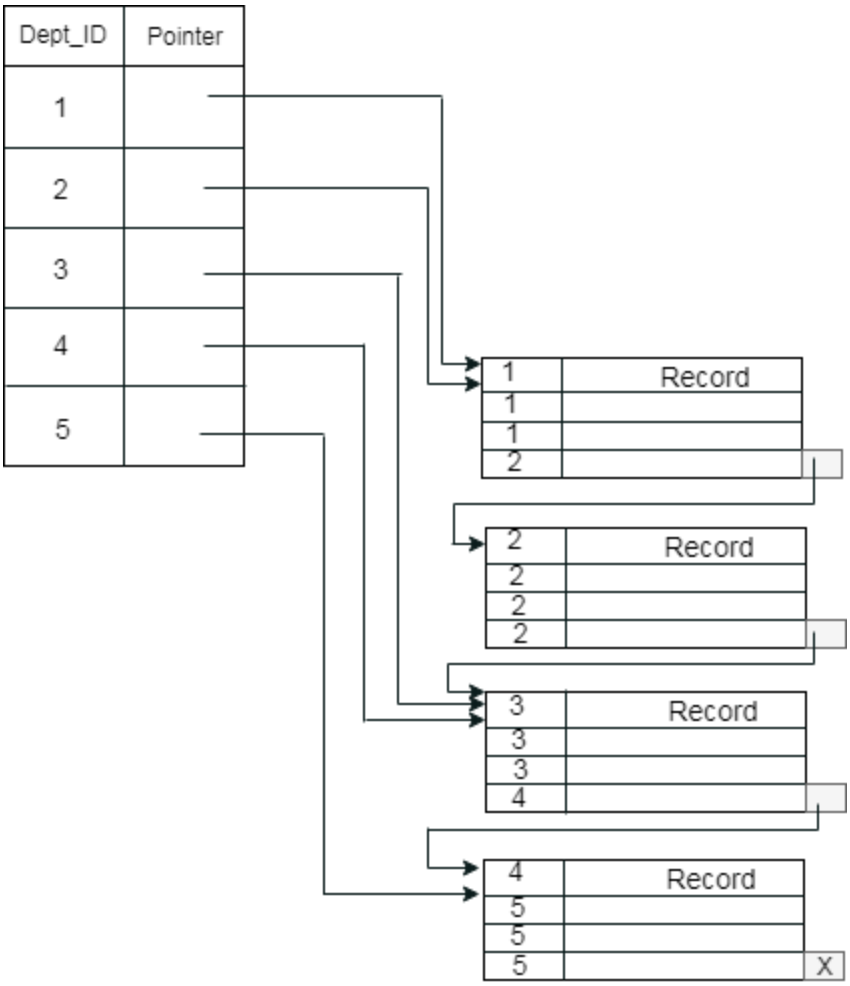
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



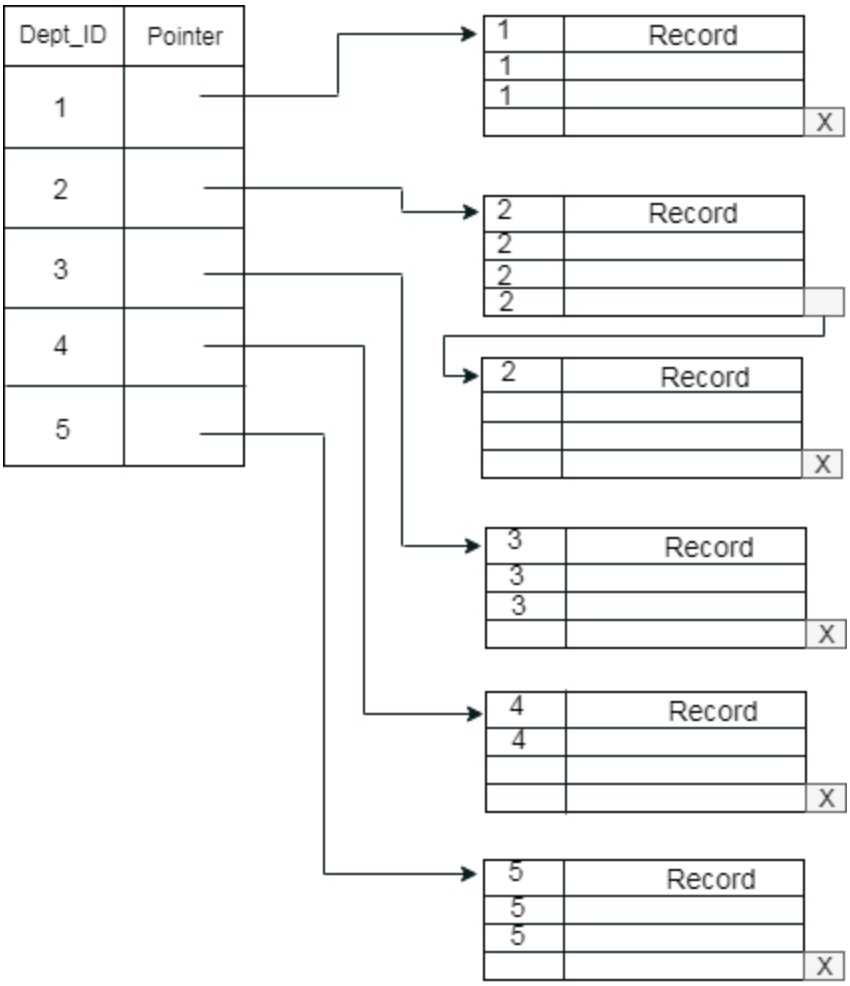
Clustering Index

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



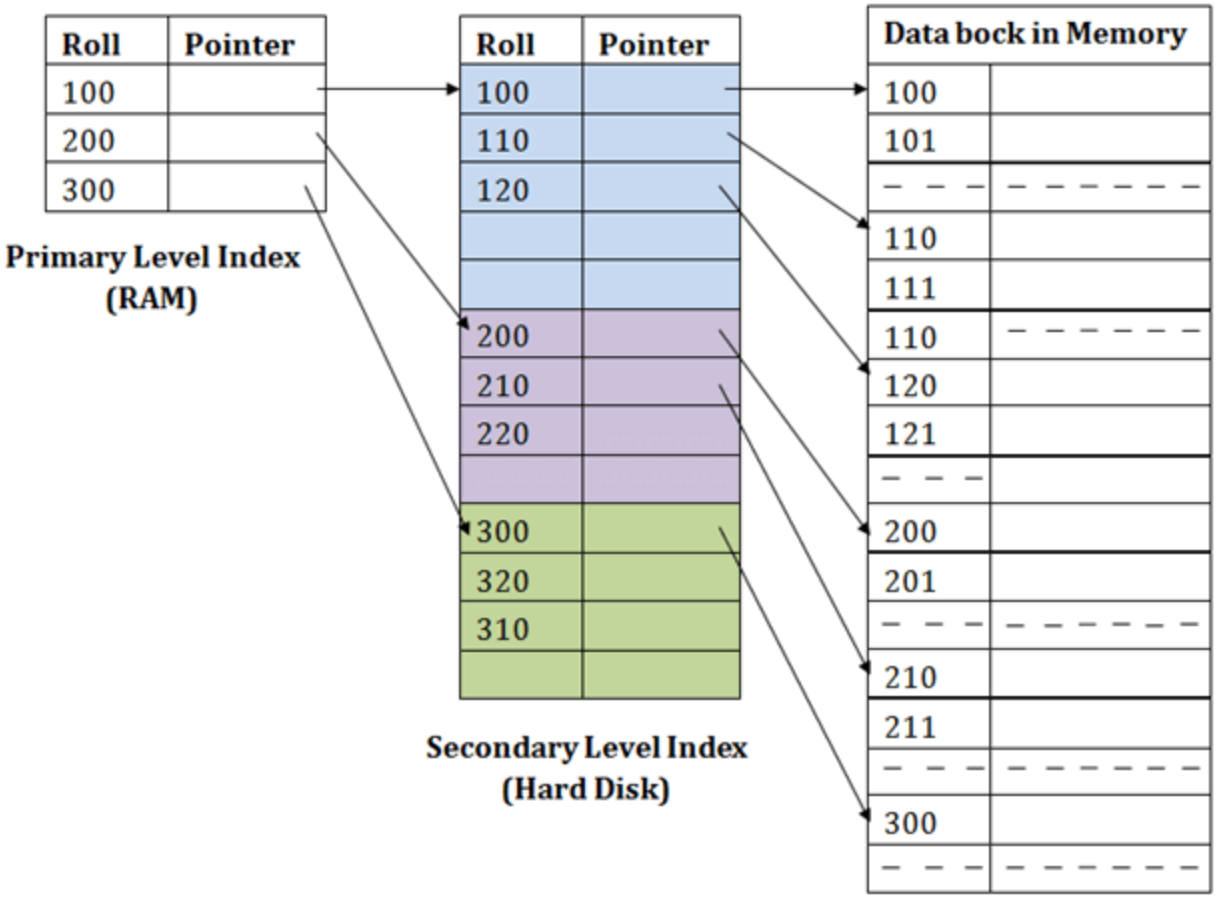
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

Next Topic B+ Tree