

UNIT 5

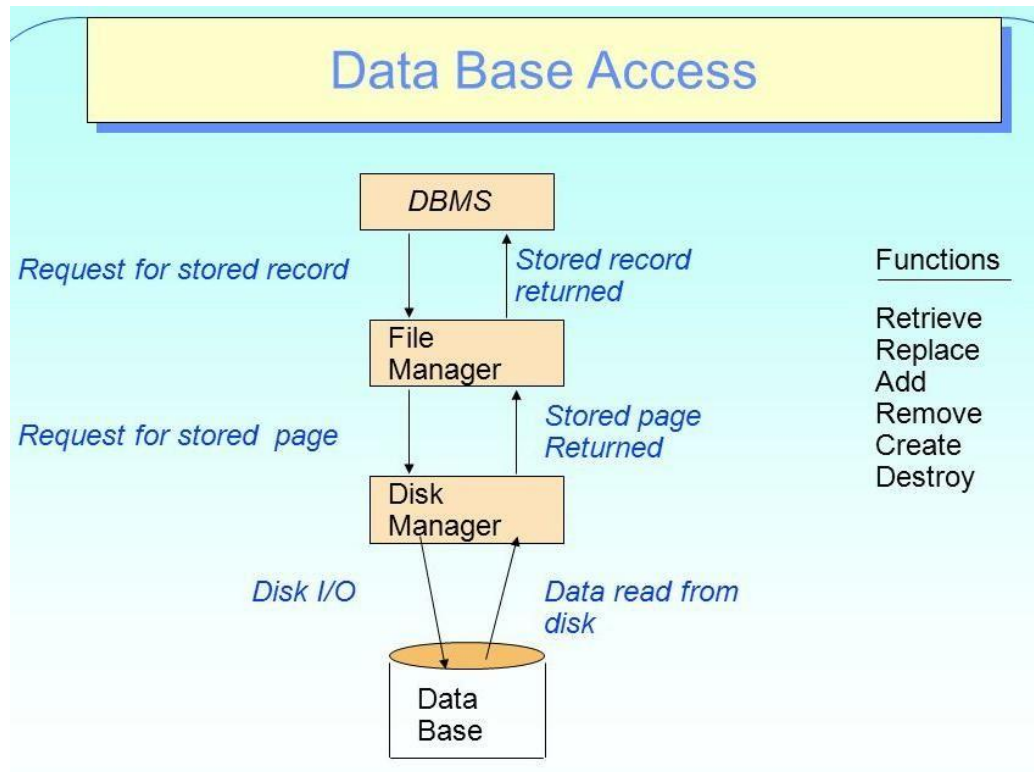
OVERVIEW OF STORAGE AND INDEXING

Unit 5 Contents at a glance:

1. Data on external storage,
2. file organizations
3. indexing,
4. index data structures,
5. comparison of file organizations,
6. RAID
7. Tree structured indexing -intuition for tree indexes,
8. indexed sequential access method (ISAM),
9. B+ Trees -a dynamic tree structure.

1. Data on external storage:

- Data in a DBMS is stored on storage devices such as disks and tapes
- The disk space manager is responsible for keeping track of available disk space.
- The file manager, which provides the abstraction of a file of records to higher levels of DBMS code, issues requests to the disk space manager to obtain and relinquish space on disk.



Storage Manager Component :

A Storage Manager is a component or program module that provides the interface between the low-level data stored in the database and the application programs/queries submitted to the system. The Storage Manager Components include –

1. **File Manager-** File manager manages the file space and it takes care of the structure of the file. It manages the allocation space on disk storage and the data structures used to represent info stored on other media.
2. **Buffer Manager** – It transfers blocks between disk (or other devices) and Main Memory. A DMA (Direct Memory Access) is a form of Input/Output that controls the exchange of blocks process. When a processor receives a request for a transfer of a block, it sends it to the DMA Controller which transfers the block uninterrupted.
3. **Authorization and Integrity Manager** – This Component of storage manager checks for the authority of the users to access and modify information, as well as integrity constraints (keys, etc).
4. **Disk Manager-** The block requested by the file manager is transferred by the Disk Manager.

Memory Hierarchy:

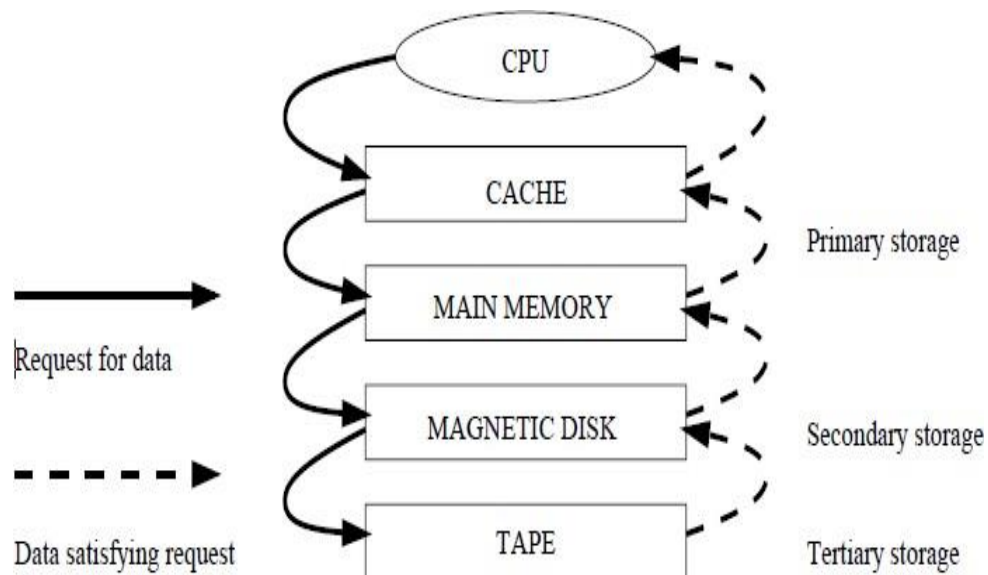


Figure: Memory Hierarchy

At the top, we have primary storage, which consists of cache and main memory, and provides very fast access to data. Then comes secondary storage, which consists of slower devices such as magnetic disks. Tertiary storage is the slowest class of storage devices; for example, optical disks and tapes.

Primary Storage:

1. all the primary storage level, the memory hierarchy includes at the most expensive end cache memory, which is a static RAM (Random Access Memory) cache memory is mainly used by the CPU to speedup execution programs.
2. the next level of primary storage is DRAM (Dynamic Random Access Memory), which provides the main work area for the CPU for keeping programs and data , which is popularly called as main memory .
3. the advantages of DRAM is its low cost, which continuous to decrease ; the drawback is its volatility and lower speed compared with static RAM.

Secondary Storage:

At the secondary storage level, the hierarchy includes magnetic disks, as well storage in the form of CD - ROM (Compact Disk - Read Only Memory) devices.

Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

Tertiary storage:

At the tertiary storage level, the hierarchy includes optical disks and tapes as the least expensive end.

The storage capacity anywhere in the hierarchy is measured in kilobytes (k bytes or bytes), megabytes (M bytes or 1 million bytes), gigabytes (G byte or billion bytes), and even terabytes (1000 G bytes).

Explanation:**DRAM:**

programs reside execute in **DRAM** . Generally, large permanent database reside on secondary storage, and portions of the database are read into and written from buffers in main memory as needed. personal computers and work stations have tens of megabytes of data in DRAM. it is become possible to load a large fraction of the database into main memory. an example is telephone switching applications, which store databases that contain routing and line information in main memory.

Flash Memory:

1. Between **DRAM and magnetic disk storage**, another form of memory resides, **flash memory**, which is becoming common, particularly because it is non - volatile.
2. flash memories are high density, high - performance memories using EEPROM (Electrically Erasable programmable Read -only Memory) technology.
3. the advantage of flash memory is the fast access speed;
4. the disadvantage is that an entire block must be erased and written over at a time.

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.

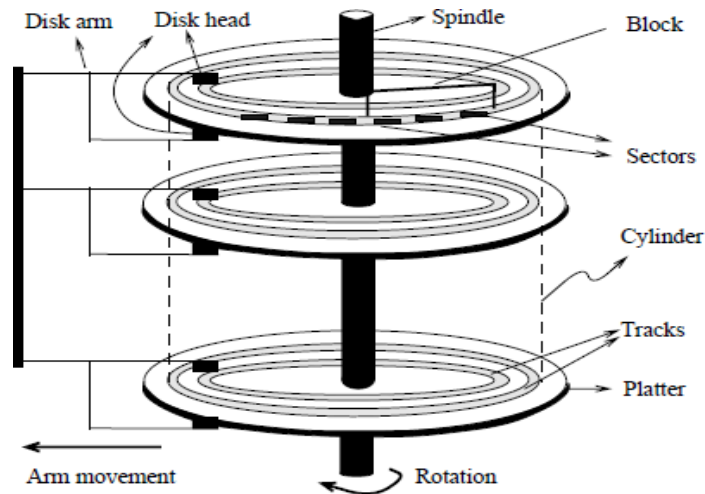


Figure 7.2 Structure of a Disk

Figure: Structure of magnetic disk

Access time: the time it takes from when a read or write request is issued to when data transfer begins.

Data-transfer rate– the rate at which data can be retrieved from or stored to the disk.

Mean time to failure (MTTF)– the average time the disk is expected to run continuously without any failure.

CD-ROM:

CD - ROM disks store data optically and are read by a laser. CD - ROM s contain pre - recorded data that cannot be overwritten. WORM (Write - Once - Read - Many disks) are a form of optical storage used for archiving data; they allow data to be written once and read any number of times without the possibility of erasing. the DVD (Digital Video Disks) is a recent standard for optical disks allowing fourteen to fifteen gigabytes of storage per disks.

Tapes:

1. Tapes are relatively not expensive and can store very large amount of data. when we maintain data for a long period but do expect to access it very often.
2. used primarily for backup and archival data.
3. Cheaper, but much slower access, since tape must be read sequentially from the beginning.
4. Used as protection from disk failures!
5. A Quantum DLT 4000 drive is a typical tape device; it stores 20 GB of data and can store about twice as much by compressing the data.

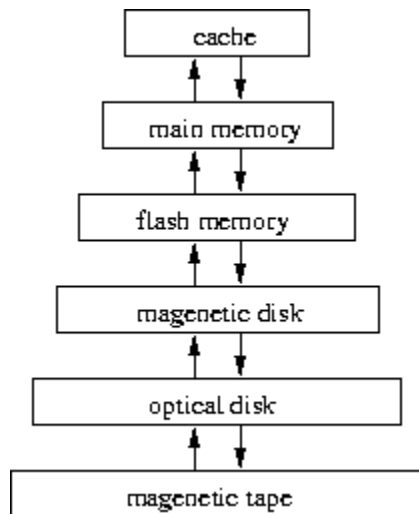


Figure: storage device hierarchy

2. File Organizations:

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

Some of the file organizations are

1. Sequential File Organization
2. Heap File Organization
3. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization
6. Cluster File Organization

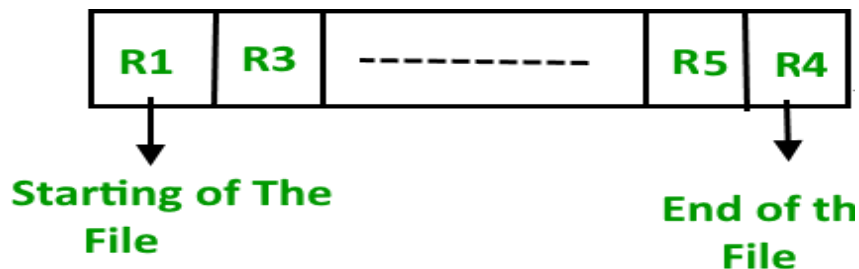
1. Sequential File Organization:

In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

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:

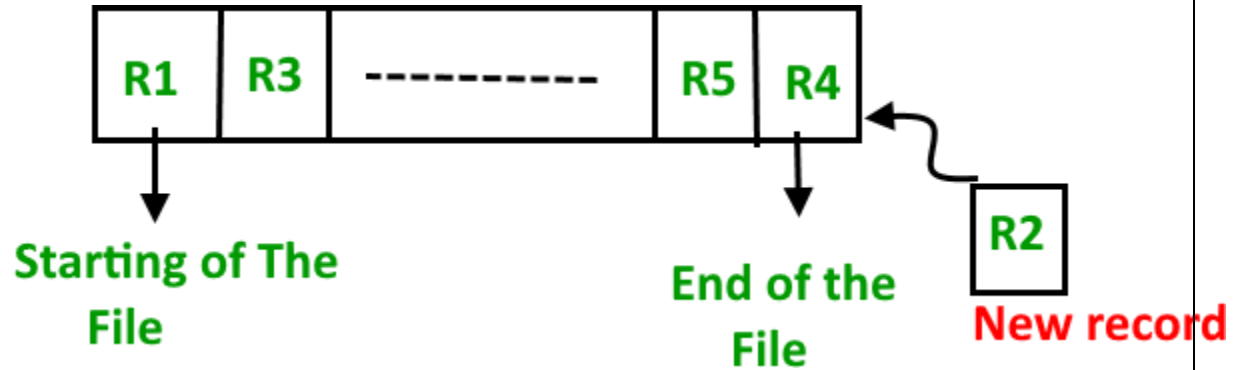
1. Pile File Method
2. Sorted File

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.

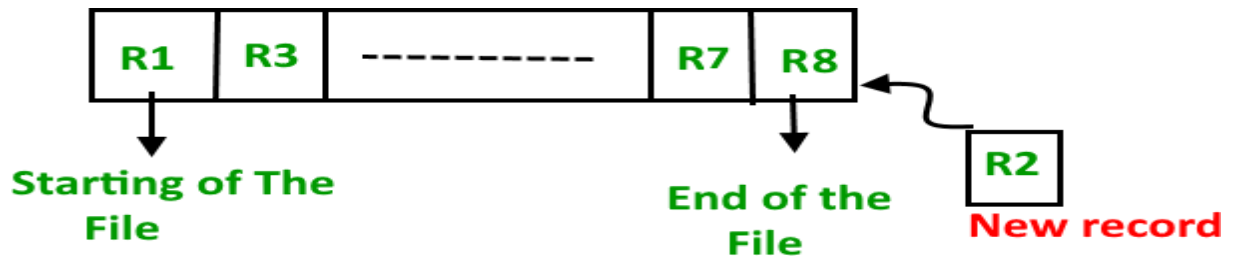


Insertion of new record –

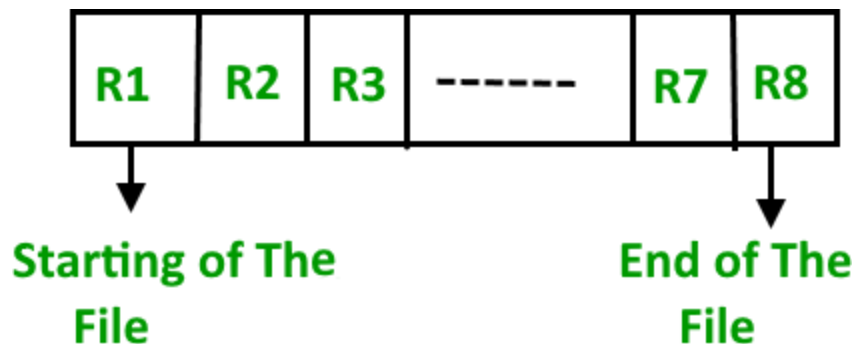
Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



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.

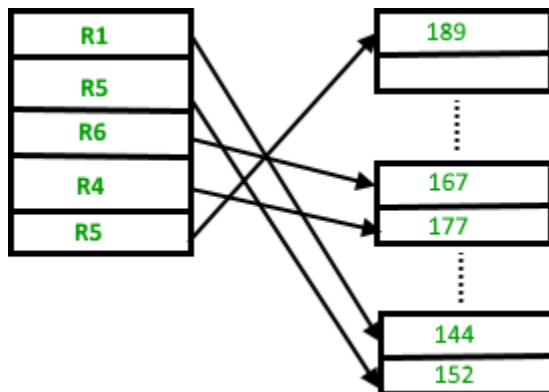
Cons –

- 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.

2. Heap File Organization:

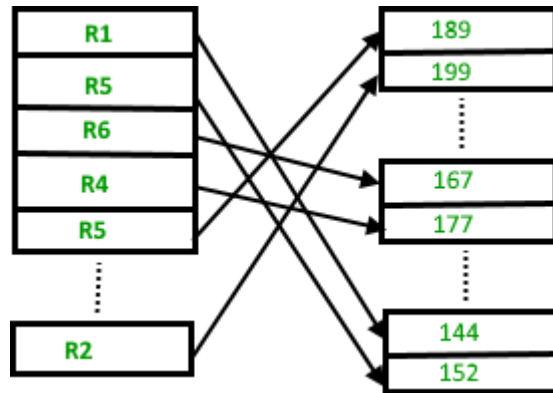
When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area.

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 then 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.

3. Hash File Organization:

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

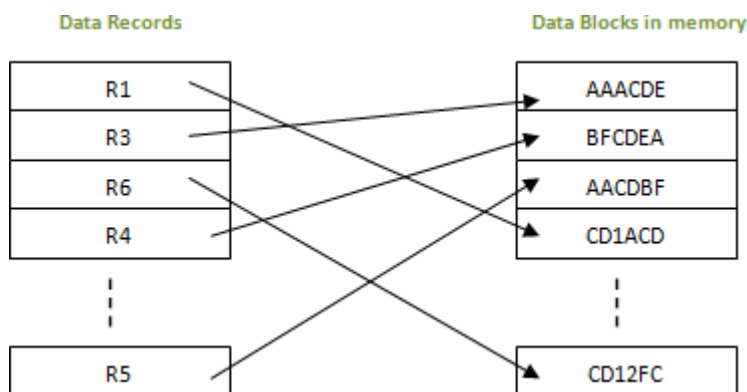
In this method of file organization, hash function is used to calculate the address of the block to store the records.

The hash function can be any simple or complex mathematical function.

The hash function is applied on some columns/attributes – either key or non-key columns to get the block address.

Hence each record is stored randomly irrespective of the order they come. Hence this method is also known as Direct or Random file organization.

If the hash function is generated on key column, then that column is called hash key, and if hash function is generated on non-key column, then the column is hash column.



When a record has to be retrieved, based on the hash key column, the address is generated and directly from that address whole record is retrieved. Here no effort to traverse through whole file. Similarly when a new record has to be inserted, the address is generated by hash key and record is directly inserted. Same is the case with update and delete.

Advantages of Hash File Organization

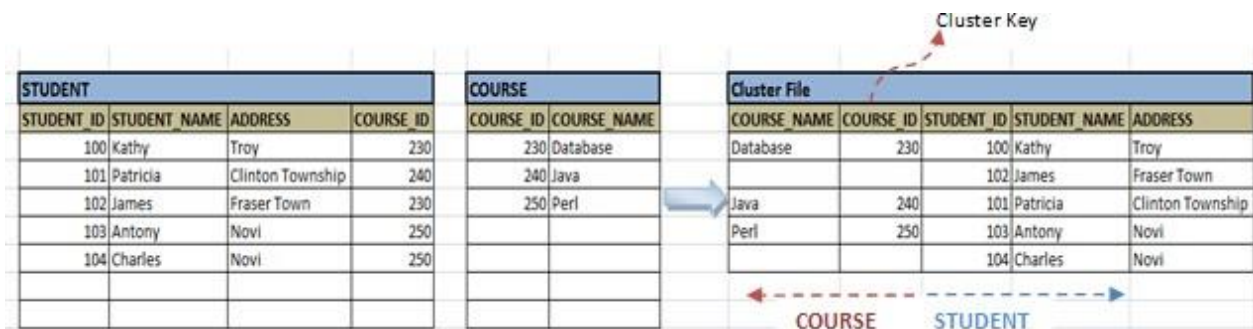
- Records need not be sorted after any of the transaction. Hence the effort of sorting is reduced in this method.
- Since block address is known by hash function, accessing any record is very faster. Similarly updating or deleting a record is also very quick.

- This method can handle multiple transactions as each record is independent of other. i.e.; since there is no dependency on storage location for each record, multiple records can be accessed at the same time.
- It is suitable for online transaction systems like online banking, ticket booking system etc.

clustered file organization:

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

In this method two or more table which are frequently used to join and get the results are stored in the same file called clusters. These files will have two or more tables in the same data block and the key columns which map these tables are stored only once. This method hence reduces the cost of searching for various records in different files. All the records are found at one place and hence making search efficient.



REDUNDANT ARRAY OF INDEPENDENT DISKS(RAID)

RAID or **R**edundant **A**rray of **I**ndependent **D**isks, is a technology to connect multiple secondary storage devices and use them as a single storage media.

RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.

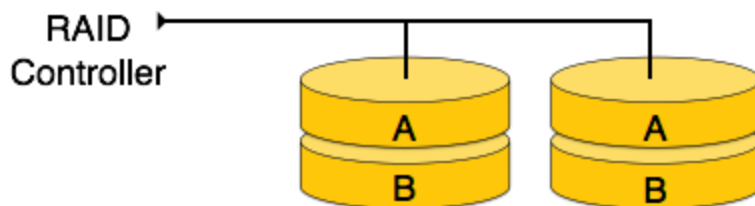
RAID 0

In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.



RAID 1

RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called **mirroring** and provides 100% redundancy in case of a failure.



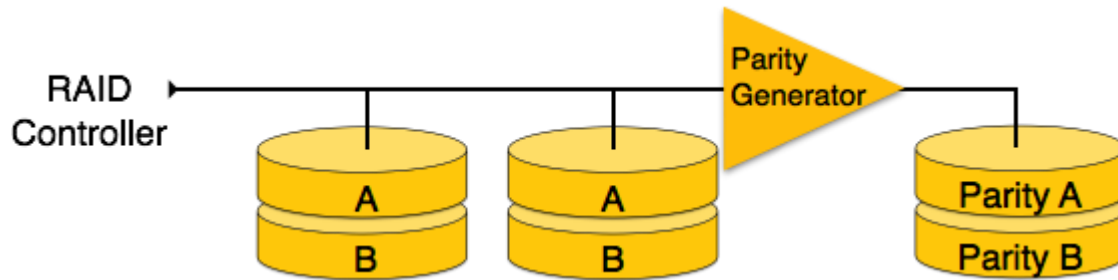
RAID 2

RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID2 is not commercially available.



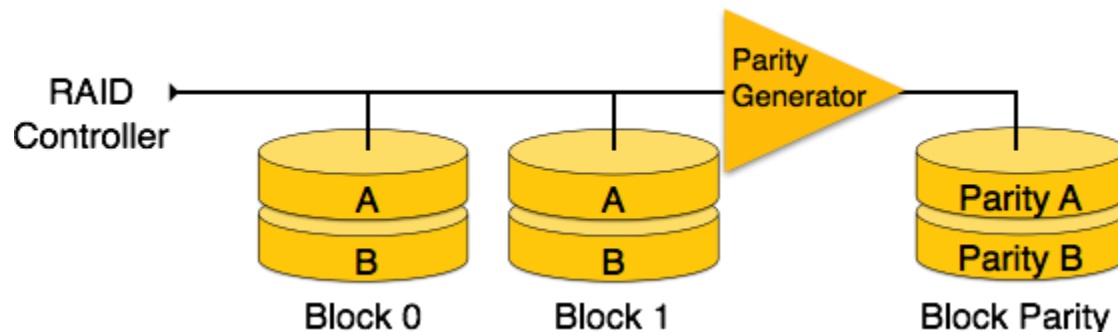
RAID 3

RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



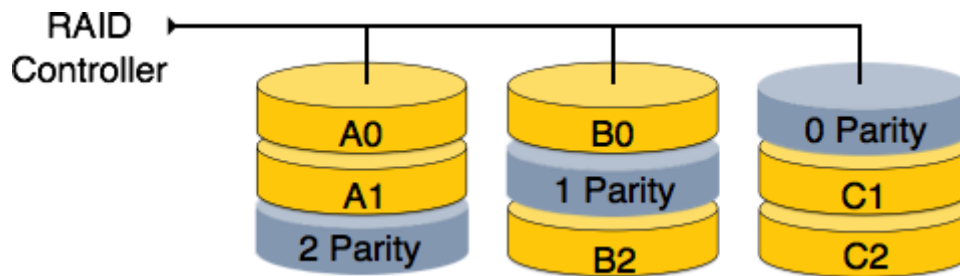
RAID 4

In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



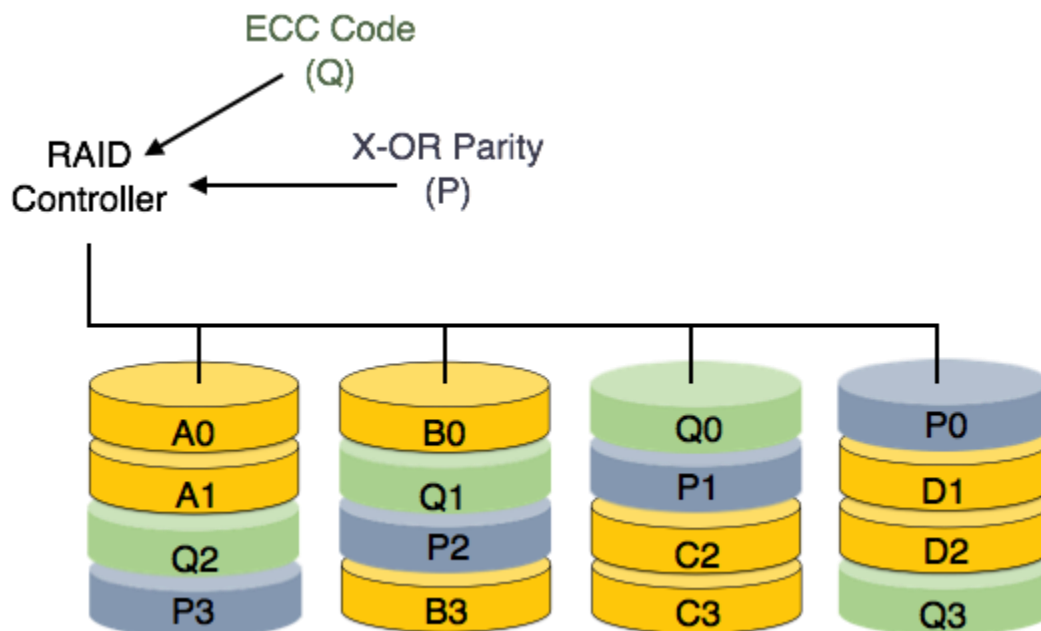
RAID 5

RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID 6

RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.



DBMS INDEXING

We know that information in the DBMS files is stored in form of records. Every record is equipped with some key field, which helps it to be recognized uniquely.

Indexing is defined based on its indexing attributes. Indexing can be one of the following types:

- **Primary Index:** If index is built on ordering 'key-field' of file it is called Primary Index. Generally it is the primary key of the relation.
- **Secondary Index:** If index is built on non-ordering field of file it is called Secondary Index.
- **Clustering Index:** If index is built on ordering non-key field of file it is called Clustering Index.

Ordering field is the field on which the records of file are ordered. It can be different from primary or candidate key of a file.

Ordered Indexing is of two types:

- Dense Index
- Sparse Index

Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index record contains search key value and a pointer to the actual record on the disk.



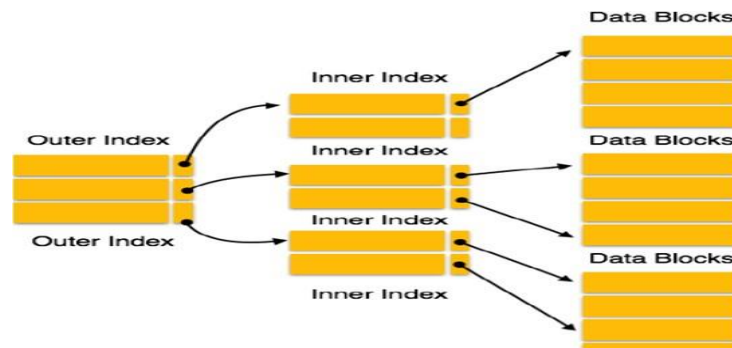
Sparse Index

In sparse index, index records are not created for every search key. An index record here contains search key and actual pointer to the data on the disk. To search a record we first proceed by index record and reach at the actual location of the data.



Multilevel Index

Index records are comprised of search-key value and data pointers. This index itself is stored on the disk along with the actual database files. As the size of database grows so does the size of indices.



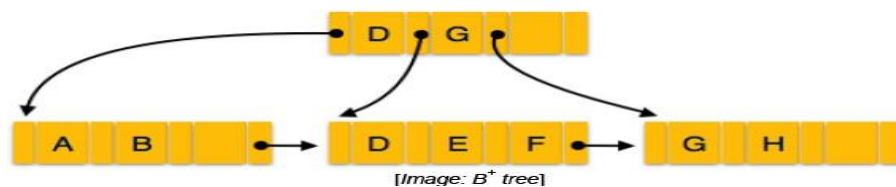
Multi-level Index helps breaking down the index into several smaller indices in order to make the outer most level so small that it can be saved in single disk block which can easily be accommodated anywhere in the main memory.

B+ TREE

B+tree is multi-level index format, which is balanced binary search trees. As mentioned earlier single level index records becomes large as the database size grows, which also degrades performance. All leaf nodes of B+ tree denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, all leaf nodes are linked using link list, which makes B+ tree to support random access as well as sequential access.

STRUCTURE OF B+ TREE

Every leaf node is at equal distance from the root node. A B+ tree is of order n where n is fixed for every B+ tree.



Internal nodes:

- ☑ Internal (non-leaf) nodes contains at least $\lceil n/2 \rceil$ pointers, except the root node.
- ☐ At most, internal nodes contain n pointers.

Leaf nodes:

- ☐ Leaf nodes contain at least $\lceil n/2 \rceil$ record pointers and $\lceil n/2 \rceil$ key values
- ☐ At most, leaf nodes contain n record pointers and n key values
- ☐ Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

B+ tree insertion

- ☐ B+ tree are filled from bottom. And each node is inserted at leaf node.

☐ If leaf node overflows:

- o Split node into two parts
- o Partition at $i = \lfloor (m+1)/2 \rfloor$
- o First i entries are stored in one node
- o Rest of the entries ($i+1$ onwards) are moved to a new node
- o i th key is duplicated in the parent of the leaf

☐ If non-leaf node overflows:

- o Split node into two parts
- o Partition the node at $i = \lfloor (m+1)/2 \rfloor$
- o Entries upto i are kept in one node
- o Rest of the entries are moved to a new node

B+ tree deletion

- ☑ B+ tree entries are deleted leaf nodes.
- ☐ The target entry is searched and deleted.
 - o If it is in internal node, delete and replace with the entry from the left position.

- ☐ After deletion underflow is tested o If underflow occurs
 - ☐ Distribute entries from nodes left to it. o If distribution from left is not possible
 - ☐ Distribute from nodes right to it
 - o If distribution from left and right is not possible
- ☐ Merge the node with left and right to it.

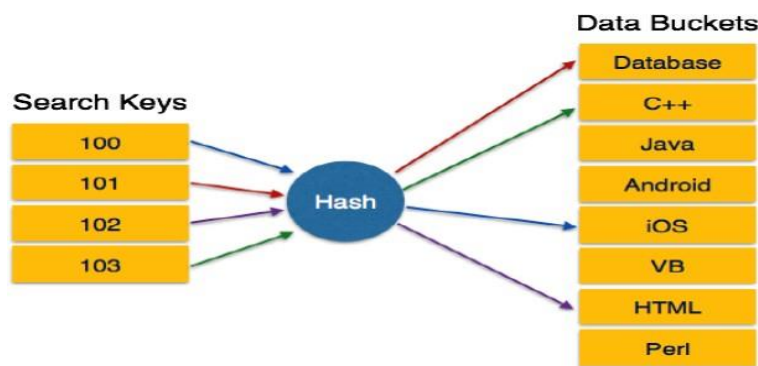
DBMS HASHING

For a huge database structure it is not sometime feasible to search index through all its level and then reach the destination data block to retrieve the desired data. Hashing is an effective technique to calculate direct location of data record on the disk without using index structure. **Hash Organization**

□ **Bucket:** Hash file stores data in bucket format. Bucket is considered a unit of storage. 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 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.



[Image: Static Hashing]

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 = $h(K)$

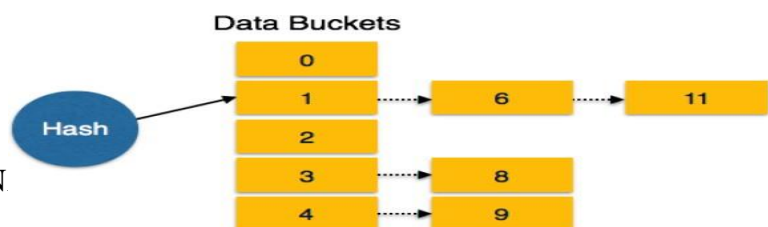
□ **Search:** When a record needs to be retrieved the same hash function can be used to retrieve the address of bucket where the data is stored.

□ **Delete:** This is simply search followed by 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.



[Image: Overflow chaining]

☐ **Linear Probing:** When 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.

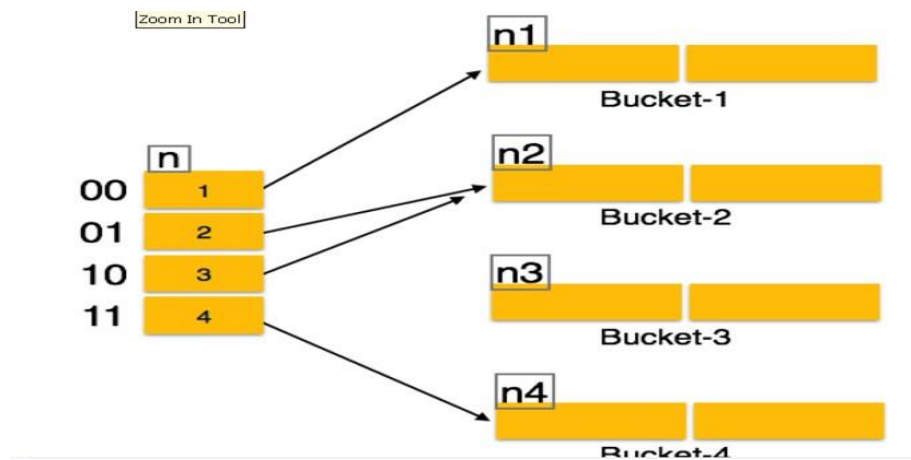


For a hash function to work efficiently and effectively the following must match:

- ☐ Distribution of records should be uniform
- ☐ Distribution should be random instead of any ordering

Dynamic Hashing

Problem with static hashing is that it does not expand or shrink dynamically as the size of 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.

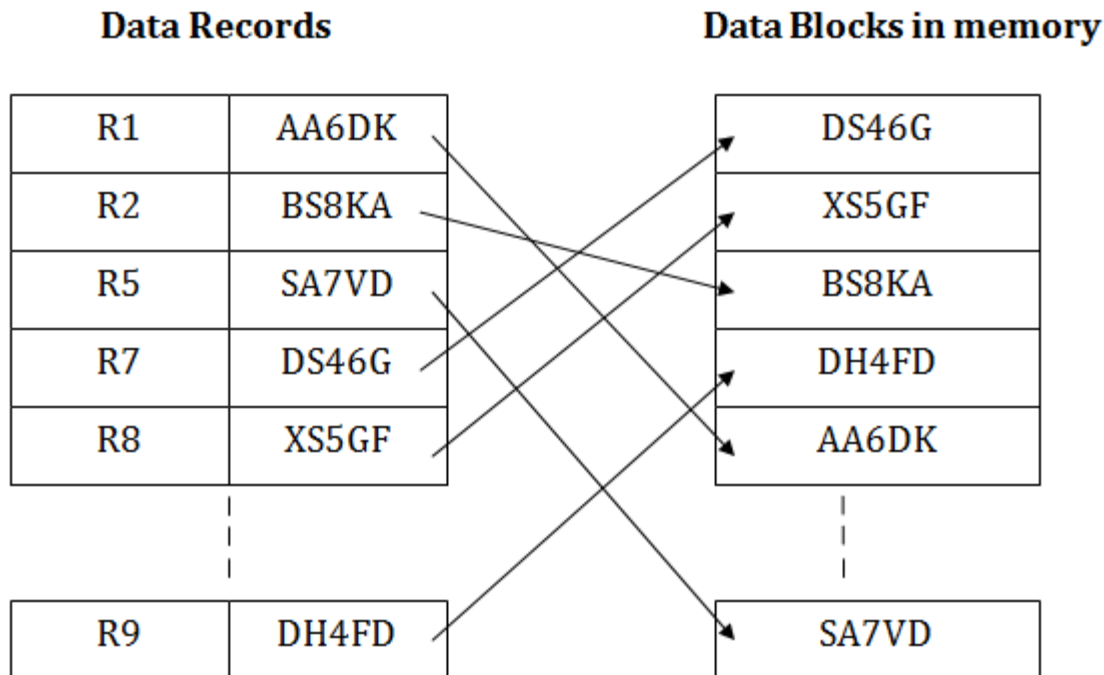


Operation

- ☐ **Querying:** Look at the depth value of hash index and use those bits to compute the bucket address.
 - ☐ **Update:** Perform a query as above and update data.
 - ☐ **Deletion:** Perform a query to locate desired data and delete data.
 - ☐ **Insertion:** compute the address of bucket o If the bucket is already full
 - ☐ Add more buckets
 - ☐ Add additional bit to hash value
 - ☐ Re-compute the hash function o Else
 - ☐ Add data to the bucket o If all buckets are full, perform the remedies of static hashing.
- Hashing is not favorable when the data is organized in some ordering and queries require range of data. When data is discrete and random, hash performs the best. Hashing algorithm and implementation have high complexity than indexing. All hash operations are done in constant time.

Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

