UNIT-IV

Overview of Storage System in DBMS

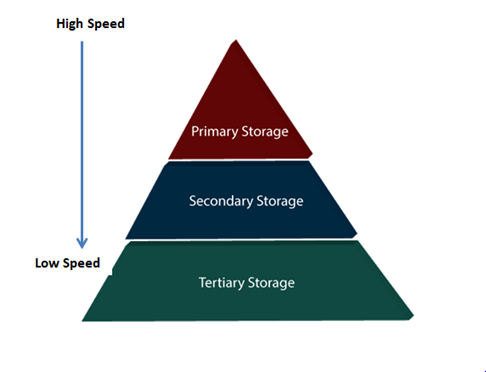
A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices.

In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

Types of Data Storage

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

* Primary Storage
* Secondary Storage
* Tertiary Storage



Primary Storage

It is the primary area that offers quick access to the stored data. We also know the primary storage as volatile storage. It is because this type of memory does not permanently store the data. As soon as the system leads to a power cut or a crash, the data also get lost. Main memory and cache are the types of primary storage.

* **Main Memory:** It is the one that is responsible for operating the data that is available by the storage medium. The main memory handles each instruction of a computer machine. This type of memory can store gigabytes of data on a system but is small enough to carry the entire database. At last, the main memory loses the whole content if the system shuts down because of power failure or other reasons.

1. **Cache:** It is one of the costly storage media. On the other hand, it is the fastest one. A cache is a tiny storage media which is maintained by the computer hardware usually. While designing the algorithms and query processors for the data structures, the designers keep concern on the cache effects.

Secondary Storage

Secondary storage is also called as Online storage. It is the storage area that allows the user to save and store data permanently. This type of memory does not lose the data due to any power failure or system crash. That's why we also call it non-volatile storage.

There are some commonly described secondary storage media which are available in almost every type of computer system:

* **Flash Memory:** A flash memory stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system. These USB keys help transfer data to a computer system, but it varies in size limits. Unlike the main memory, it is possible to get back the stored data which may be lost due to a power cut or other reasons. This type of memory storage is most commonly used in the server systems for caching the frequently used data. This leads the systems towards high performance and is capable of storing large amounts of databases than the main memory.
* **Magnetic Disk Storage:** This type of storage media is also known as online storage media. A magnetic disk is used for storing the data for a long time. It is capable of storing an entire database. It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing. Also, if the system performs any operation over the data, the modified data should be written back to the disk. The tremendous capability of a magnetic disk is that it does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data.

Tertiary Storage

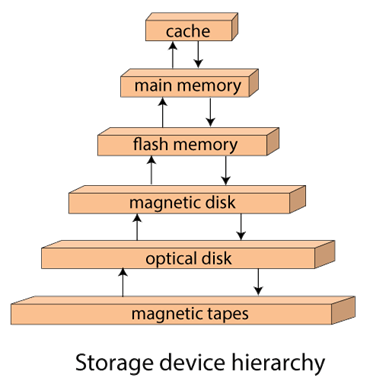
It is the storage type that is external from the computer system. It has the slowest speed. But it is capable of storing a large amount of data. It is also known as Offline storage. Tertiary storage is generally used for data backup. There are following tertiary storage devices available:

* **Optical Storage:** An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.
* **Tape Storage:** It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.

Storage Hierarchy

Besides the above, various other storage devices reside in the computer system. These storage media are organized on the basis of data accessing speed, cost per unit of data to buy the medium, and by medium's reliability. Thus, we can create a hierarchy of storage media on the basis of its cost and speed.

Thus, on arranging the above-described storage media in a hierarchy according to its speed and cost, we conclude the below-described image:



In the image, the higher levels are expensive but fast. On moving down, the cost per bit is decreasing, and the access time is increasing. Also, the storage media from the main memory to up represents the volatile nature, and below the main memory, all are non-volatile devices.

RAID

RAID refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used.

It consists of an array of disks in which multiple disks are connected to achieve different goals.

RAID technology

There are 7 levels of RAID schemes. These schemas are as RAID 0, RAID 1, ...., RAID 6.

These levels contain the following characteristics:

* It contains a set of physical disk drives.
* In this technology, the operating system views these separate disks as a single logical disk.
* In this technology, data is distributed across the physical drives of the array.
* Redundancy disk capacity is used to store parity information.
* In case of disk failure, the parity information can be helped to recover the data.

Standard RAID levels

RAID 0

* RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
* This level doesn't provide fault tolerance but increases the system performance.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** |
| 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 |

In this figure, block 0, 1, 2, 3 form a stripe.

In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one.

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** |
| 20 | 22 | 24 | 26 |
| 21 | 23 | 25 | 27 |
| 28 | 30 | 32 | 34 |
| 29 | 31 | 33 | 35 |

In this above figure, there is no duplication of data. Hence, a block once lost cannot be recovered.

Pros of RAID 0:

* In this level, throughput is increased because multiple data requests probably not on the same disk.
* This level full utilizes the disk space and provides high performance.
* It requires minimum 2 drives.

Cons of RAID 0:

* It doesn't contain any error detection mechanism.
* The RAID 0 is not a true RAID because it is not fault-tolerance.
* In this level, failure of either disk results in complete data loss in respective array.

RAID 1

This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** |
| A | A | B | B |
| C | C | D | D |
| E | E | F | F |
| G | G | H | H |

Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

Pros of RAID 1:

* The main advantage of RAID 1 is fault tolerance. In this level, if one disk fails, then the other automatically takes over.
* In this level, the array will function even if any one of the drives fails.

Cons of RAID 1:

* In this level, one extra drive is required per drive for mirroring, so the expense is higher.

RAID 2

* RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.
* Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

Pros of RAID 2:

* This level uses one designated drive to store parity.
* It uses the hamming code for error detection.

Cons of RAID 2:

* It requires an additional drive for error detection.

RAID 3

* RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
* In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
* In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** |
| A | B | C | P(A, B, C) |
| D | E | F | P(D, E, F) |
| G | H | I | P(G, H, I) |
| J | K | L | P(J, K, L) |

Pros of RAID 3:

* In this level, data is regenerated using parity drive.
* It contains high data transfer rates.
* In this level, data is accessed in parallel.

Cons of RAID 3:

* It required an additional drive for parity.
* It gives a slow performance for operating on small sized files.

RAID 4

* RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
* This level allows recovery of at most 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
* Level 3 and level 4 both are required at least three disks to implement RAID.

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** |
| A | B | C | P0 |
| D | E | F | P1 |
| G | H | I | P2 |
| J | K | L | P3 |

In this figure, we can observe one disk dedicated to parity.

In this level, parity can be calculated using an XOR function. If the data bits are 0,0,0,1 then the parity bits is XOR(0,1,0,0) = 1. If the parity bits are 0,0,1,1 then the parity bit is XOR(0,0,1,1)= 0. That means, even number of one results in parity 0 and an odd number of one results in parity 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **C1** | **C2** | **C3** | **C4** | **Parity** |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |

Suppose that in the above figure, C2 is lost due to some disk failure. Then using the values of all the other columns and the parity bit, we can recompute the data bit stored in C2. This level allows us to recover lost data.

RAID 5

* RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.
* It consists of block-level striping with DISTRIBUTED parity.
* Same as RAID 4, this level allows recovery of at most 1 disk failure. If more than one disk fails, then there is no way for data recovery.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Disk 0** | **Disk 1** | **Disk 2** | **Disk 3** | **Disk 4** |
| 0 | 1 | 2 | 3 | P0 |
| 5 | 6 | 7 | P1 | 4 |
| 10 | 11 | P2 | 8 | 9 |
| 15 | P3 | 12 | 13 | 14 |
| P4 | 16 | 17 | 18 | 19 |

This figure shows that how parity bit rotates.

This level was introduced to make the random write performance better.

Pros of RAID 5:

* This level is cost effective and provides high performance.
* In this level, parity is distributed across the disks in an array.
* It is used to make the random write performance better.

Cons of RAID 5:

* In this level, disk failure recovery takes longer time as parity has to be calculated from all available drives.
* This level cannot survive in concurrent drive failure.

RAID 6

* This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
* In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5, and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk 1** | **Disk 2** | **Disk 3** | **Disk 4** |
| A0 | B0 | Q0 | P0 |
| A1 | Q1 | P1 | D1 |
| Q2 | P2 | C2 | D2 |
| P3 | B3 | C3 | Q3 |

Pros of RAID 6:

* This level performs RAID 0 to strip data and RAID 1 to mirror. In this level, stripping is performed before mirroring.
* In this level, drives required should be multiple of 2.

Cons of RAID 6:

* It is not utilized 100% disk capability as half is used for mirroring.
* It contains very limited scalability.

File Organization Storage

There are different ways of storing data in the database. Storing data in files is one of them. A user can store the data in files in an organized manner. These files are organized logically as a sequence of records and reside permanently on disks. Each file is divided into fixed-length storage units known as **Blocks**. These blocks are the units of storage allocation as well as data transfer. Although the default block size in the database is 4 to 8 kilobytes, many databases allow specifying the size at the time of creating the database instance.

Usually, the record size is smaller than the block size. But, for large data items such as images, the size can vary. For accessing the data quickly, it is required that one complete record should reside in one block only. It should not be partially divided between one or two blocks. In RDBMS, the size of tuples varies in different relations. Thus, we need to structure our files in multiple lengths for implementing the records. In file organization, there are two possible ways of representing the records:

* Fixed-length records
* Variable-length records

Let's discuss this in detail.

Fixed-Length Records

Fixed-length records means setting a length and storing the records into the file. If the record size exceeds the fixed size, it gets divided into more than one block. Due to the fixed size there occurs following two problems:

1. Partially storing subparts of the record in more than one block requires access to all the blocks containing the subparts to read or write in it.
2. It is difficult to delete a record in such a file organization. It is because if the size of the existing record is smaller than the block size, then another record or a part fills up the block.

However, including a certain number of bytes is the solution to the above problems. It is known as **File Header**. The allocated file header carries a variety of information about the file, such as the address of the first record. The address of the second record gets stored in the first record and so on. This process is similar to pointers. The method of insertion and deletion is easy in fixed-length records because the space left or freed by the deleted record is exactly similar to the space required to insert the new records. But this process  fails for storing the records of variable lengths.

Variable-Length Records

Variable-length records are the records that vary in size. It requires the creation of multiple blocks of multiple sizes to store them. These variable-length records are kept in the following ways in the database system:

1. Storage of multiple record types in a file.
2. It is kept as Record types that enable repeating fields like multisets or arrays.
3. It is kept as Record types that enable variable lengths either for one field or more.

In variable-length records, there exist the following two problems:

1. Defining the way of representing a single record so as to extract the individual attributes easily.
2. Defining the way of storing variable-length records within a block so as to extract that record in a block easily.

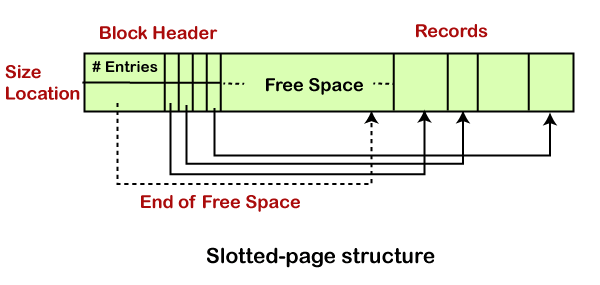
Thus, the representation of a variable-length record can be divided into two parts:

1. An initial part of the record with fixed-length attributes such as numeric values, dates, fixed-length character attributes for storing their value.
2. The data for variable-length attributes such as varchar type is represented in the initial part of the record by (offset, length) pair. The offset refers to the place where that record begins, and length refers to the length of the variable-size attribute. Thus, the initial part stores fixed-size information about each attribute, i.e., whether it is the fixed-length or variable-length attribute.

Slotted-page Structure

There occurs a problem to store variable-length records within the block. Thus, such records are organized in a slotted-page structure within the block. In the slotted-page structure, a header is present at the starting of each block. This header holds information such as:

1. The number of record entries in the header
2. No free space remaining in the block
3. An array containing the information on the location and size of the records.



**Inserting and Deleting Method**

The variable-length records reside in a contiguous manner within the block.

When a new record is to be inserted, it gets the place at the end of the free space. It is because free space is contiguous as well. Also, the header fills an entry with the size and location information of the newly inserted record.

When an existing record is deleted, space is freed, and the header entry sets to deleted. Before deleting, it moves the record and occupies it to create the free space. The end-of-free-space gets the update. Then all the free space again sets between the first record and the final entry.

The primary technique of the slotted-page structure is that no pointer should directly point the record. Instead, it should point to the header entry that contains the information of its location. This stops fragmentation of space inside the block but supports indirect pointers to the record.

File Organization

* The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
* File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
* File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
* The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
* Files of fixed length records are easier to implement than the files of variable length records.

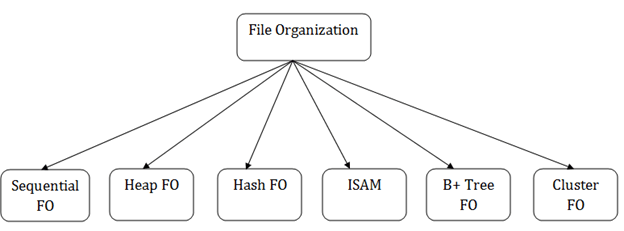
Objective of file organization

* It contains an optimal selection of records, i.e., records can be selected as fast as possible.
* To perform insert, delete or update transaction on the records should be quick and easy.
* The duplicate records cannot be induced as a result of insert, update or delete.
* For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



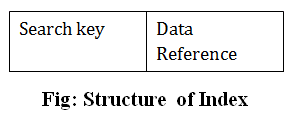
* [Sequential file organization](https://www.javatpoint.com/dbms-sequential-file-organization)
* [Heap file organization](https://www.javatpoint.com/dbms-heap-file-organization)
* [Hash file organization](https://www.javatpoint.com/dbms-hash-file-organization)
* [B+ file organization](https://www.javatpoint.com/dbms-b-plus-file-organization)
* [Indexed sequential access method (ISAM)](https://www.javatpoint.com/dbms-indexed-sequential-access-method)
* [Cluster file organization](https://www.javatpoint.com/dbms-cluster-file-organization)

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.

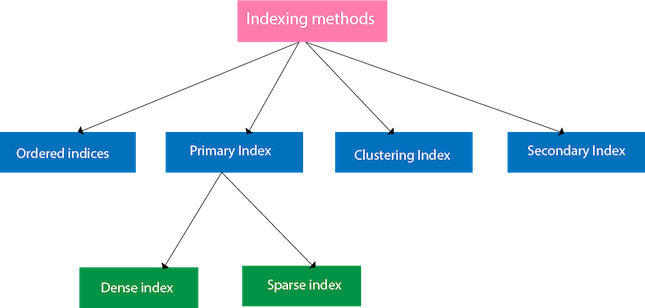
Index structure:

Indexes can be created using some database columns.



* 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.

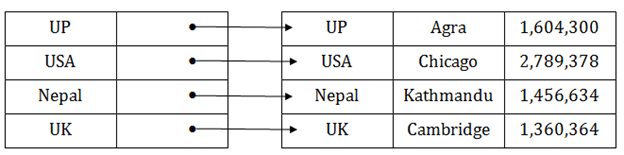
* In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543\*10=5430 bytes.
* In the case of an index, we will search using indexes and the DBMS will read the record after reading 542\*2= 1084 bytes which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

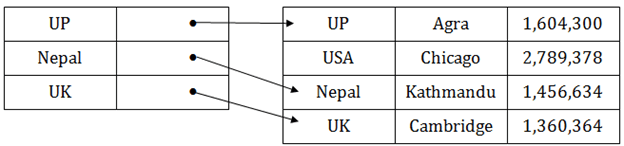
Dense index

* 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

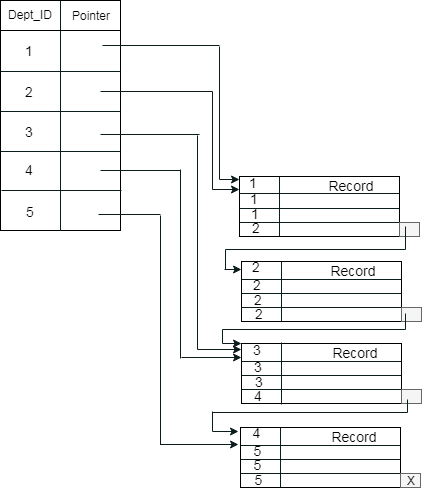
* In the data file, index record appears only for a few items. Each item points to a block.
* In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



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.



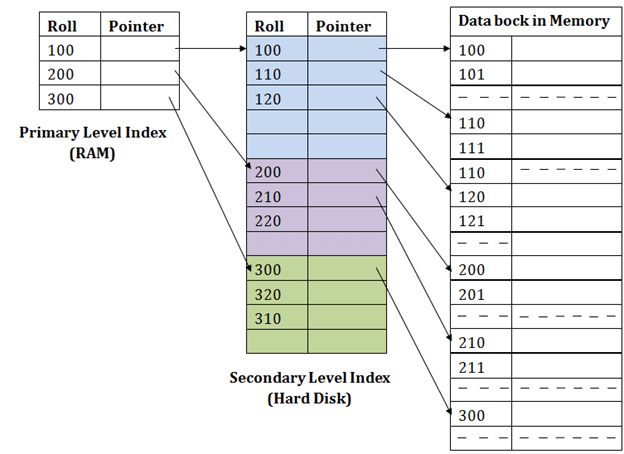
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



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:**

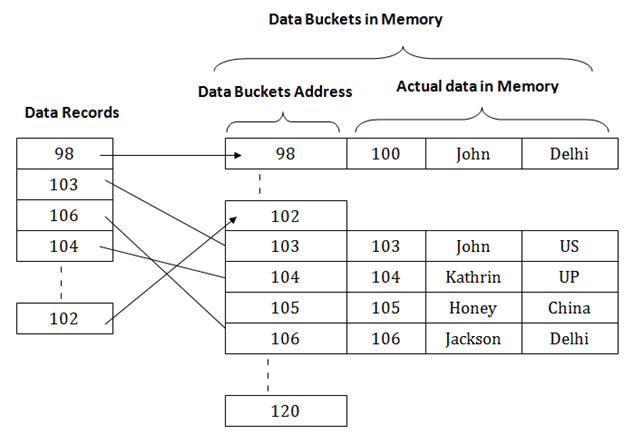
* If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
* Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it 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.

Hashing

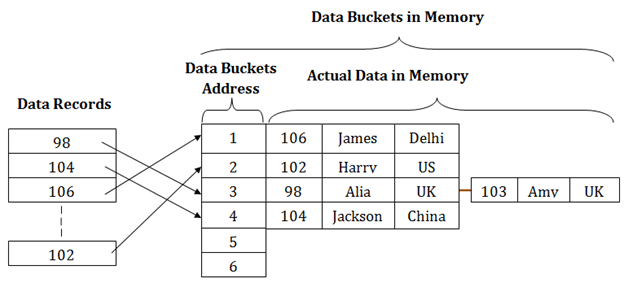
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

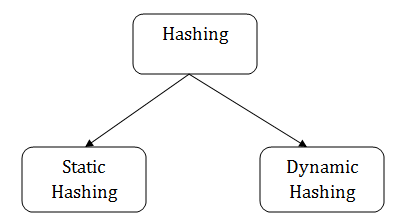
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



Types of Hashing:

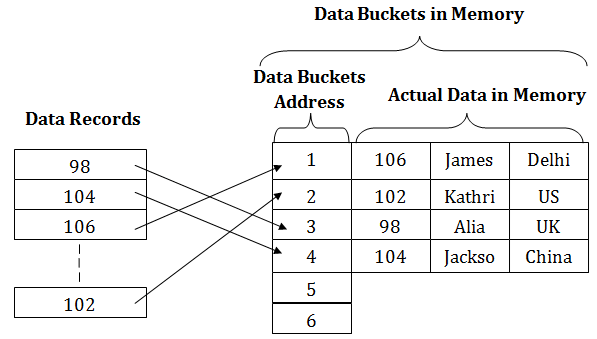


* [Static Hashing](https://www.javatpoint.com/dbms-static-hashing)
* [Dynamic Hashing](https://www.javatpoint.com/dbms-dynamic-hashing)

Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP\_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



Operations of Static Hashing

* **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

* **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

* **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

* **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

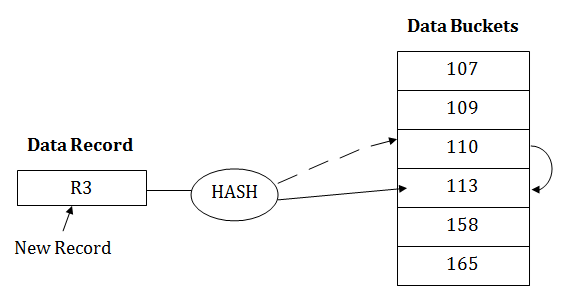
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

1. Open Hashing

When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.

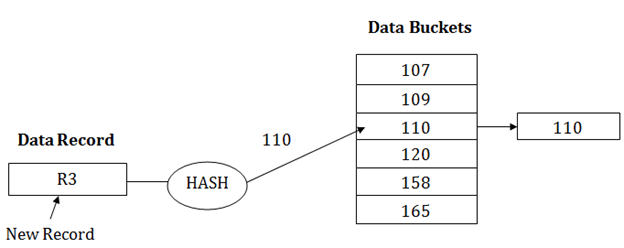
**For example:** suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



2. Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

**For example:** Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

* The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
* In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
* This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

How to search a key

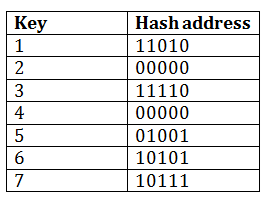
* First, calculate the hash address of the key.
* Check how many bits are used in the directory, and these bits are called as i.
* Take the least significant i bits of the hash address. This gives an index of the directory.
* Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

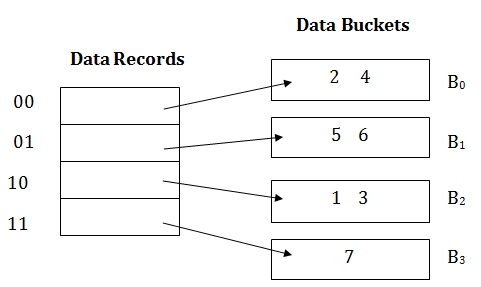
* Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
* If there is still space in that bucket, then place the record in it.
* If the bucket is full, then we will split the bucket and redistribute the records.

For example:

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

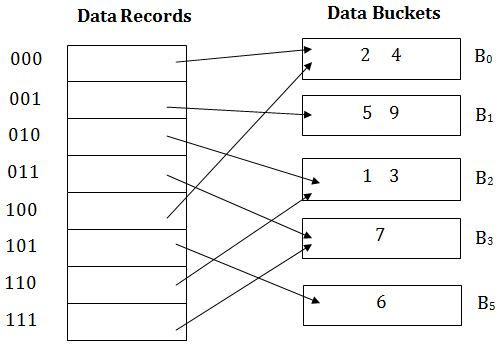


The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

* Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
* The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
* Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
* Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
* Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing

* In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
* In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
* This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

* In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
* In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.