# 1. Introduction to Hashing

Hashing is presented as an efficient alternative to indexing for the fast retrieval of records from a database based on a search key. It operates by using a hash function to compute the address of the disk block containing the desired record, significantly reducing access times.

# 2. Internal Hashing

Internal hashing is used for managing files or records stored in RAM. A hash table, implemented as an array of records, uses a common hash function (**h(k) = K mod M**) to determine the array index for storing records. Collisions, where two records compute to the same index, are addressed through methods like open addressing, chaining, and multiple hashing.

# 3. External Hashing

For disk-based files, external hashing uses a hashing function to assign records to buckets, which are either individual disk blocks or clusters of contiguous blocks. The method ensures a minimal number of disk accesses, typically one, to retrieve a record.

# 4. Static External Hashing

In static external hashing, the number of buckets is fixed, which poses challenges for dynamically sized files. The system uses chaining to handle bucket overflows, but this can lead to performance issues as the file grows and collisions become more frequent.

# 5. Dynamic Hashing

To overcome the limitations of static hashing, dynamic hashing allows the hash address space to expand or shrink dynamically. It utilizes a binary pattern for hash values and distributes records based on the leading bits of these values.

## Extendible Hashing

A form of dynamic hashing that uses a directory structure to access buckets efficiently. It adapts to the file's growth without significant performance degradation, allocating additional buckets as needed.

## Linear Hashing

Another dynamic approach that does not rely on a directory structure. Instead, it uses multiple hash functions to split overflowing buckets and redistribute records. This method manages collisions and bucket expansions in a linear and orderly manner.