# Reflection 2

## Advanced Database Management System

# Reflection Topics

- Seek Time
- Latency
- Buffering of Blocks
- File Organization
- Parallelizing Disk Access using RAID Technology
- Hashing
- Single level Indexes
- B Trees
- B+ Trees

# Seek Time

- The amount of time taken by the read/write head of the disc to move from one part of the disk to another is called the seek time.
- The seek time can differ for the same disc.
- Seek time is generally measured as an average seek time.

# Rotational Delay

- Also called Latency.
- The amount of time taken from the beginning of the desired track to rotate to the required position for the read/write head.

# Block Transfer Time

- Additional time needed to transfer the data.

- Total time needed to locate and transfer an arbitrary block, given its address is:

Total response time = seek time + latency
+ block transfer time

# Buffering of Blocks

- A buffer is a reserved area of the main memory which holds one block.
- Several buffers can be reserved in memory to speed up the transfer.
- While one buffer is being read or written, the CPU can process data in the other buffer. This technique is called double buffering.
- Double buffering can improve disk access performance.
- In double buffering an independent disk I/O processor, once started, can proceed to transfer a block between a buffer and disk independent of and in parallel to CPU processing.

# File Organization

- Sorted Files
  - Ordered/Sequential Files
- Heap Files
  - Unordered/Pile Files

# Sorted Files

- Also called **ordered** or **sequential** files.
- The value of **Ordering Field** is used to sort the file records.
- Reading the records in order of the ordering field is quite efficient.
- Insertion is expensive because records must be inserted in the correct order.
- A **binary search** can be used to search for a record on its *ordering field* value.
  - This requires reading and searching $\log_2$ of the file blocks on the average, an improvement over linear search.

# Heap Files

- Also called a **Unordered** or a **pile** file.
- New records are always inserted at the end of the file.
- Reading the file records in order of a particular field first requires sorting.
- Searching the file records is quite expensive.
- A **linear search** is used to search for a file record.
  - This requires reading and searching half the file blocks on the average.
- Record insertion is quite efficient.

# RAID Technology – Concept and Usage

- **RAID -** Redundant Arrays of Inexpensive Disks.
- The main aim of RAID is to even out the widely different rates of performance improvement of disks against those in memory and microprocessors.
- **Data striping** is used to utilize parallelism to improve disk performance.
- Data striping distributes data transparently over multiple disks and make them appear as a single large and fast disk.

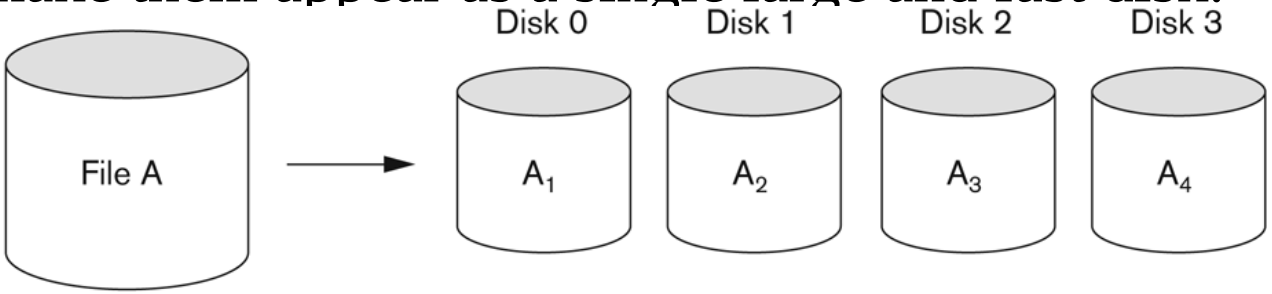



Fig. : Data Striping – File A is stripped across four disks.

# RAID Levels

- RAID level 0 has no redundant data and hence has the best write performance at the risk of data loss.
- RAID level 1 uses mirrored disks.
- RAID level 2 uses memory-style redundancy by using Hamming codes, which contain parity bits for distinct overlapping subsets of components. Level 2 includes both error detection and correction.
- RAID level 3 uses a single parity disk relying on the disk controller to figure out which disk has failed.
- RAID Levels 4 and 5 use block-level data striping, with level 5 distributing data and parity information across all disks.
- RAID level 6 applies the so-called P + Q redundancy scheme using Reed-Soloman codes to protect against up to two disk failures by using just two redundant disks.

# Hashing Techniques

- Used to organize files for very fast access of records and the organized files are called Hash Files.
- The field on which search (equality) condition is applied is called Hash Field.
- If the hash filed is a key field then the field is called Hash Key.
- The function applied on a hash field is called Hash (Randomizing) function.
- Hash Function yields the address of the disk block in which the record is stored.

# Internal Hashing

- Hashing for internal files is called internal hashing.
- In this, hashing is typically implemented as a Hash Table through the use of an array of records.
- Hash address is generated by the following methods:
  - By applying the mod hash function: A hash function h(K) is selected to transforms the hash field value (K) into an integer between 0 and M-1 and used to address the record.

$$h(K) = K \bmod M$$

  - Folding technique: Arithmetic or logical functions are applied to different portions of hash field value to calculate hash address.

# Collision & Its Resolution

- A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record.
- Collision Resolution:
  - Open Addressing
  - Chaining
  - Multiple Hashing
- Chaining algorithms are easy to implement.
- Deletion algorithms for open addressing are tricky.

# External Hashing

- Hashing of disk files is called **external hashing**.
- The target address space is made of buckets, numbered $bucket_0$, $bucket_1$, ..., $bucket_{Max-1}$.
  - A bucket corresponds to one disk block or a cluster of contiguous blocks.
- One of the file fields is chosen to be the **hash key** of the file.
- The record with hash key value K is stored in bucket i, where i=h(K), and h is the **hashing function**.
- Search is very efficient on the hash key but expensive for a record given a value of record other than hash field .

# Single-level Indexes

- Indexing mechanisms is used to speed up the retrieval of desired data from a file.
- Categories of Indexes
  - Dense Index: It has an index entry for every search key value in the data file.
  - Sparse Index: It is also called non-dense index. This index has index entries for only some of the search values.
- Types of single-level indexes
  - Primary Index
  - Clustered Index
  - Secondary Index

# Primary Indexes

- A primary index is an ordered file whose records are of fixed length with two fields; primary key and a pointer to a disk block (a disk address).
- The first record in each block of the data file is called anchor record of the block or the block anchor.
- Index file for a primary index needs substantially fewer blocks than the data file.
- A primary index is a non-dense (sparse) index.

# Clustering Index

- Clustering index is also an ordered file with two fields; first field is same as of clustering field (field that does not have a distinct value for the each record) of the data file, and second field is a block pointer.
- It is used to speed up the retrieval of records of a file that have same value for the clustering field (non-key field).
- Clustering index is also a non-dense (sparse) index

# Secondary Indexes

- Secondary Index provides a secondary means of accessing a file for which some primary access already exists.
- Secondary index is an ordered file with two fields; indexing field and a block pointer or a record pointer.
- Secondary index is a dense index.
- Secondary index usually needs more storage space and longer search time than a primary index.

# B-Tree

- B-tree is a specialized multiway tree designed especially for use on disk.
- B-Tree consists of a root node, branch nodes and leaf nodes containing the indexed field values in the ending (or leaf) nodes of the tree.

# B-Tree Characteristics

- In a B-tree each node may contain a large number of keys
- B-tree is designed to branch out in a large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small
- Constraints that tree is always balanced
- Space wasted by deletion, if any, never becomes excessive
- Insert and deletions are simple processes
  - Complicated only under special circumstances

    -Insertion into a node that is already full or a deletion from a node makes it less then half full

# Characteristics of a B-Tree of Order P

- Within each node, $K_1 < K_2 < .. < K_{p-1}$
- Each node has at most p tree pointer
- Each node, except the root and leaf nodes, has at least ceil(p/2) tree pointers, The root node has at least two tree pointers unless it is the only node in the tree.
- All leaf nodes are at the same level. Leaf node have the same structure as internal nodes except that all of their tree pointer $P_i$ are null.

# B-Tree Insertion

1) B-tree starts with a single root node (which is also a leaf node) at level 0.
2) Once the root node is full with p – 1 search key values and when attempt to insert another entry in the tree, the root node splits into two nodes at level 1.
3) Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes.
4) When a nonroot node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
5) If the parent node is full, it is also split.
6) Splitting can propagate all the way to the root node, creating a new level if the root is split.

# B-Tree Deletion

1) If deletion of a value causes a node to be less than half full, it is combined with it neighboring nodes, and this can also propagate all the way to the root.

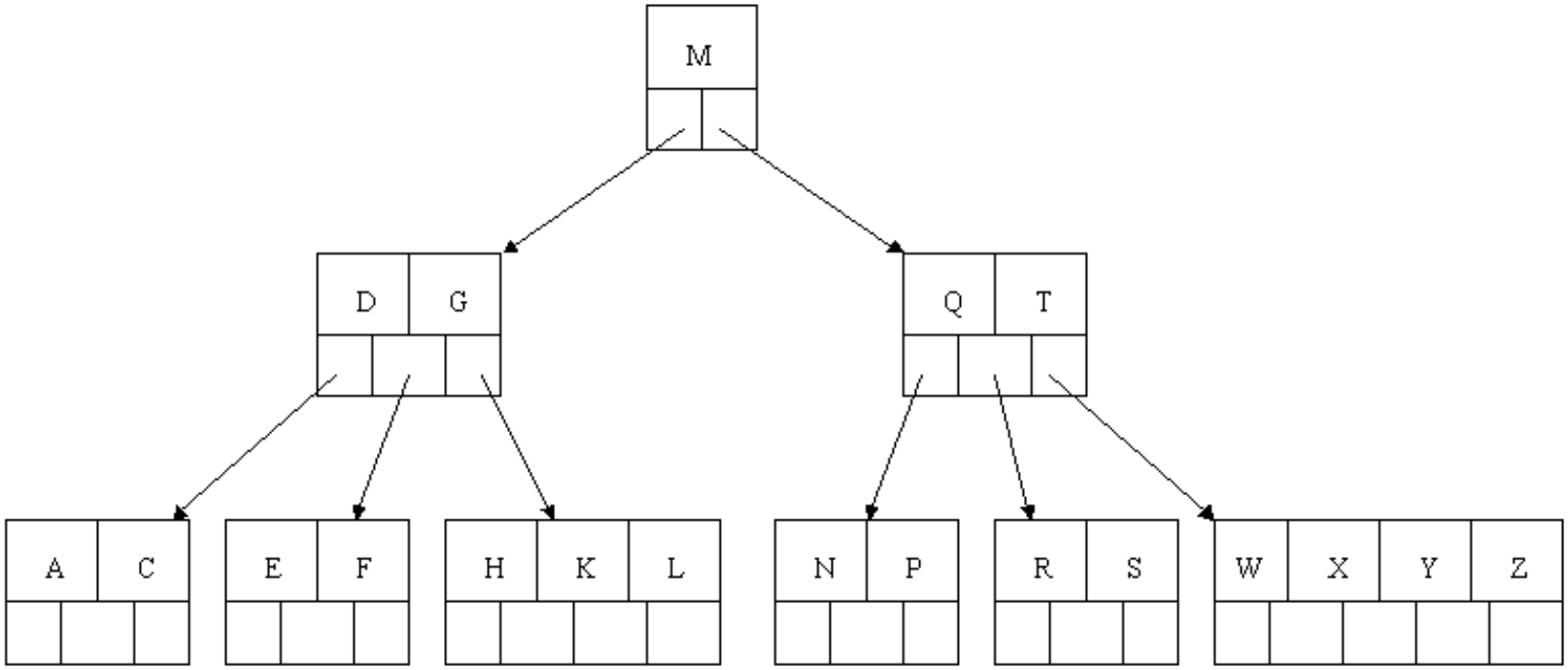
   - Can reduce the number of tree levels.

# B-tree of Order 5 Example

- All internal nodes have at least ceil(5 / 2) = ceil(2.5) = 3 children (and hence at least 2 keys), other then the root node.
- The maximum number of children that a node can have is 5 (so that 4 is the maximum number of keys)
- each leaf node must contain at least 2 keys

# B-Tree Order 5 Insertion

- Insert C N G A H E K Q M F W L T Z D P R X Y S



# B-Tree Order 5 Deletion

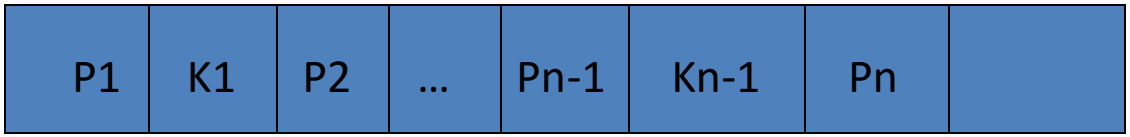
- Delete H
  - Since H is in a leaf and the leaf has more than the minimum number of keys, we just remove it.
- Delete T.
  - Since T is not in a leaf, we find its successor (the next item in ascending order), which happens to be W.
  - Move W up to replace the T. That way, what we really have to do is to delete W from the leaf .

# B+ Tree Structure

- A B+ Tree is in the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is the same length.
- Each nonleaf node in the tree has between [n/2] and n children, where n is fixed.
- B+ Trees are good for searches, but cause some overhead issues in wasted space.

- The pointer Pi can point to either a file record or a bucket of pointers which each point to a file record.

- A typical node contains up to n – 1 search key values K1, K2,…, Kn-1, and n pointers P1, P2,…, Pn. The search key values are kept in sorted order.



# B+ Tree Characteristics

- Data records are only stored in the leaves.
- Internal nodes store just keys.
- Keys are used for directing a search to the proper leaf.
- If a target key is less than a key in an internal node, then the pointer just to its left is followed.
- If a target key is greater or equal to the key in the internal node, then the pointer to its right is followed.

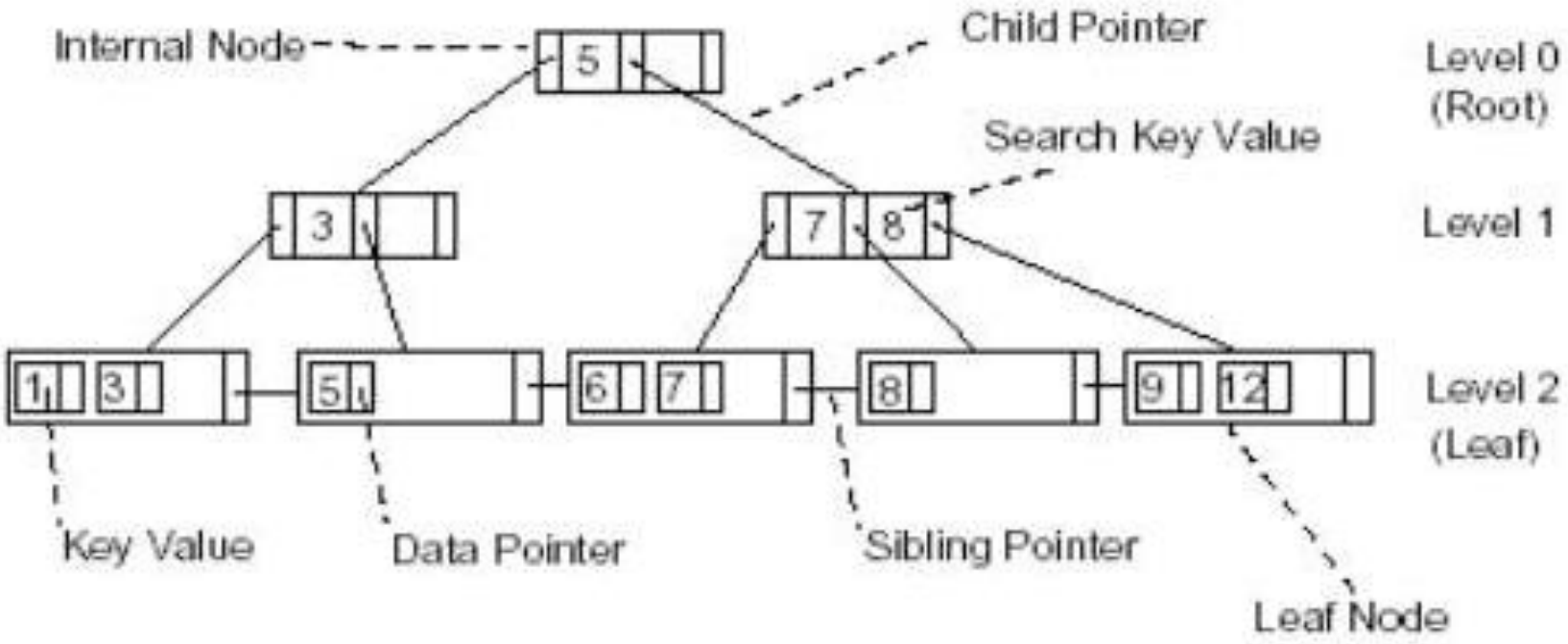
## B+- Tree Characteristics Cont.

- B+ Tree combines features of ISAM (Indexed Sequential Access Method) and B Trees.
- Very Fast Searching
- Insertion and deletion are expensive.

# Formula n-order B+ tree with a height of h

- Maximum number of keys is $n^h$
- Minimum number of keys is $2(n / 2)^{h - 1}$

# B+- Tree Structure



# B+ Tree Updates

- Insertion – If the new node has a search key that already exists in another leaf node, then it adds the new record to the file and a pointer to the bucket of pointers. If the search key is different from all others, it is inserted in order.
- Deletion – It removes the search key value from the node.

# B-Trees v/s. $B^+$-Trees

- B-Trees may use less tree nodes than a corresponding $B^+$-Tree.
- In B-Trees, sometimes possible to find search-key value before reaching leaf node.
- In B-Trees, only small fraction of all search-key values are found early
- B-Trees typically have greater depth than corresponding $B^+$-Tree
- Insertion and deletion more complicated than in $B^+$-Trees
- Implementation is harder than $B^+$-Trees.