Query Optimization? Explain in brief. [4+4]

What do you mean by RAID? Briefly explain the different levels of RAID. Explain the working of static hash indexing along with suitable examples. [1+4+3]

Define transaction and its

a) What is an index sequential file? Differentiate between dense index and sparse index. [2+2]

b) Summarize the various levels of RAID and mention how to select an appropriate level of RAID. [3+1]

[4+4]

Why does RAID Level 6 give better data protection then RAID 5? Briefly explain Fixed Length Record and Variable Length record along with suitable examples. [3+4]

a) What do you mean by hashing and indexing? Differentiate between dense index and sparse index? [2+2]

b) What do you mean by RAID? Explain the types of RAID and mention how to select an appropriate level of RAID? [4]

[5+3]

7. a) What do you mean by hashing and indexing? Differentiate between dense index and sparse index? [2+2]

b) Write about fixed length record and variable length record organization in DBMS? [4]

a) What is transaction?

a) What is the difference between ordered indices and hash indices in a database? What is the advantages of using sparse index? [2+2]

b) What do you mean by RAID? Explain the types of RAID and mention how to select an appropriate level of RAID. [1+3]

a) What are the possible transaction

a) Describe about fixed-length record and variable length record along with examples. [4]

b) Describe B+ tree structure used for indexing. [4]

a) What is the difference between ordered indices and hash indices in a database? What is the advantage of using sparse index? [2+2]

b) Write about fixed length record and variable length record organization in DBMS? [4]

Discuss about sequential file organization and multi-table clustering file organization. Explain dense index file and sparse index file. [4+4]

with suitable example.

a) What is RAID? Which RAID level would you prefer the best for safety of application and why? [1+3]

b) What is indexing? Why dynamic hashing is advantageous over static hashing? [1+3]

a) What is the use of RAID storage device? How is a record searched from a sparse sequential index? [2+3]

b) Explain about the remote backup system with diagram. [3]

6. What is RAID? Distinguish between dense and sparse indices along with example. [3+5]

a) Explain, along with an example, how a database record is searched using a sparse primary index? Write the SQL syntax to create an index. [3+1]

b) Explain the node structure of a B+ tree. Why is B+ tree good for indexing? [2+2]

a) What is the difference between ordered indices and hash indices in a database? What is the advantage of using a sparse index? [2+2]

b) What is a RAID? How would you choose the best RAID level for your database server? [1+3]

7. a) Distinguish between dense index and sparse index? What is a secondary index? [3+2]

b) Briefly explain how variable length records are stored in databases? [3]

What is the use of RAID storage device? What are the advantages and disadvantages of mirroring? [3+2]

What do you mean by ordered index and hash index? Explain limitation of static hashing. How extendable hashing overcome such limitation? [2+2+4]

7. a) Briefly explain the RAID levels 0 to 6. [6]

b) Differentiate between a dense and a sparse index. [5]

7. What is RAID? Explain the B+ tree index with an example? [3 + 8]

Overview of Physical Storage Media

1. Cache – A small, high-speed memory that stores frequently accessed data for quick retrieval.

2. Main Memory – RAM (Random Access Memory) that provides fast, temporary storage for running applications and processes.

3. Flash Memory – A non-volatile memory type used in SSDs, USB drives, and memory cards, offering faster access than magnetic disks.

4. Magnetic Disk Storage – Traditional hard drives (HDDs) that use spinning disks and magnetic heads to store data.

5. Optical Storage – CDs, DVDs, and Blu-ray discs that use laser technology for reading and writing data.

6. Tape Storage – A sequential access storage medium used for backup and archival purposes, offering high capacity but slower retrieval speeds.

RAID

-> RAID stands for Redundant Array of Independent Disks. It's a technology that combines multiple physical disk drives into one or more logical units for the purposes of data redundancy, performance improvement, or both.
-> RAID is used to protect data against hardware failures and to enhance performance. Depending on the RAID level chosen, you can optimize for speed, redundancy, or a balance of both.

Concepts:

1. Striping: Distributes data across multiple drives, increasing read/write speed.
2. Mirroring: Creates an exact copy of data on two or more disks, enhancing data security.
3. Parity: Uses calculated information (parity bits) that can be used to recover data if one drive fails.

Common RAID Levels

1. RAID 0 – Striping

-> Data is split evenly across two or more disks without any redundancy.

Advantages:

-> High Performance: Increased throughput and fast data access.

-> Efficient Storage Utilization: 100% of the disk space is used for data.

Disadvantages:

-> No Fault Tolerance: If one disk fails, all data is lost.

Use Cases:

-> Environments where speed is critical and data loss is not a major concern (e.g., temporary data processing, gaming systems).

2. RAID 1 – Mirroring

-> Data is duplicated on two or more disks. Each disk is an exact copy of the other.

Advantages:

-> High Fault Tolerance: If one drive fails, the system can continue operating using the mirror.

-> Simple Data Recovery: Easy to restore data from the surviving disk.

Disadvantages:

-> Storage Cost: Only half (or less) of the total disk capacity is usable since every piece of data is stored twice.

-> No Performance Gain on Writes: While reads can be faster if the system is optimized, write operations are duplicate

Use Cases:

-> Critical systems where data integrity is more important than storage efficiency (e.g., operating systems, databases).

(RAID 2,3,4 levels are less common in modern systems and outdated)

3. RAID 2 – Bit-Level Striping with Hamming Code

-> Data is striped at the bit level across multiple disks, with extra disks used for error correction using Hamming codes.

Pros:

-> High data integrity through error correction.

Cons:

-> Rarely used due to complexity and the availability of more efficient methods in other RAID levels.

4. RAID 3 – Byte-Level Striping with Dedicated Parity

-> Data is striped at the byte level across disks with one dedicated disk storing parity information for error correction.

Pros:

-> Can recover data if a single disk fails.

Cons:

-> The dedicated parity disk can become a performance bottleneck during intensive operations.

5. RAID 4 – Block-Level Striping with Dedicated Parity

-> Data is striped in blocks across disks, with a dedicated disk holding parity information.

Pros:

-> Fault tolerance similar to RAID 3; can rebuild data if one disk fails.

Cons:

-> Like RAID 3, the single parity disk can slow down write performance because all parity data is written to one disk.

6. RAID 5 – Striping with Parity

-> Data and parity information are striped across three or more disks. Parity allows recovery of data if one disk fails.

Advantages:

-> Provides both speed (due to striping) and fault tolerance (due to parity).

-> Efficient Storage Use: Only one disk's worth of space is used for parity regardless of the total number of disks.

Disadvantages:

-> Calculating and writing parity information can slow down write performance.

-> Rebuilding a failed disk can be time-consuming

Use Cases:

-> File and application servers where moderate performance and data redundancy are both necessary.

7. RAID 6 – Striping with Double Parity

-> Similar to RAID 5, but with an additional parity block. This allows to withstand two disk failures.

Advantages:

-> Enhanced Fault Tolerance: Can tolerate two simultaneous drive failures.

-> Data Integrity: Extra parity improves reliability in environments with larger arrays.

Disadvantages:

-> Increased Write Overhead: Additional parity calculations further slow down write performance compared to RAID 5.

-> More Complex Rebuild Process: Recovery from failure is more intensive due to double parity.

Use Cases:

-> Mission-critical applications and larger storage arrays where high data integrity and uptime are required.

8. RAID 10 (1+0) – Combining Mirroring and Striping

-> Combines the mirroring of RAID 1 with the striping of RAID 0. Data is mirrored and then striped across multiple drives.

Advantages:

-> High Performance

-> Fault Tolerance

Disadvantages

-> High Cost:

-> Complexity

Use Cases - Database systems, high-performance servers

File organization

File organization in a DBMS determines how data is stored, accessed, and managed on disk. Two common ways of organizing records are using fixed length records and variable length records.

Fixed Length Records

-> Fixed length records are records where each record occupies the same amount of space regardless of the actual data stored. Each field in the record is allocated a fixed number of bytes.

-> Every record is exactly the same size, which simplifies calculations for record positions (e.g., to jump to the Nth record, the system can compute the offset easily).

-> Faster random access is possible because the location of each record can be computed using the record size.

-> If a field is allocated more space than needed, unused space is wasted, which can lead to inefficiency.

-> The simplicity in structure makes it easier for the DBMS to manage and index the records.

Example:

Consider a student record where each record contains:

-> Student ID: 10 characters

-> Name: 30 characters (allocated fixed space, even if the name is shorter)

-> Age: 2 characters (for two-digit ages)

-> Department: 20 characters

Each record always occupies 10 + 30 + 2 + 20 = 62 characters. Even if a student's name is only 15 characters, the remaining 15 characters will remain unused.

Variable Length Records

-> Variable length records allow each record to have a different size, based on the actual length of the data in variable fields. They use pointers to mark the boundaries of fields within the record.

-> Space is used more efficiently since fields can use only as much space as required.

-> Useful when the size of data can vary significantly between records

-> Finding the start of a record or a particular field may require scanning through previous records, which can slow down random access.

-> extra metadata (such as pointers) is required to manage the variable sizes, which adds some overhead.

Example:

Consider a product review record where each record contains:

-> Review ID: Fixed 5 characters

-> Review Text: A variable length text field that could range from 20 characters to 1000 characters

-> Reviewer Name: Variable length

Organization of Records in Files

1. Heap File Organization

-> In a heap (or unordered) file organization, records are stored in no particular order. New records are simply appended to the end of the file (or placed in available free space). There is no defined ordering based on a key field.

index sequential file

2. Sequential File Organization

-> In sequential file organization, records are stored in a sorted order according to one or more key fields. This ordered arrangement facilitates efficient sequential access, which is useful for range queries.

3. Hashing File Organization

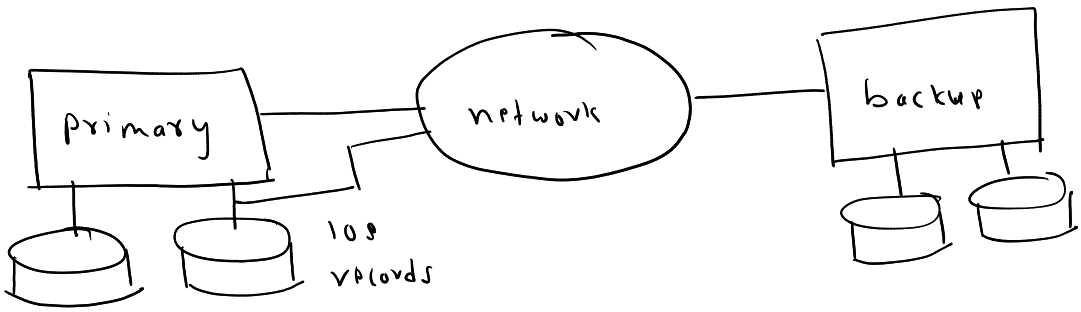
-> Hashing file organization uses a hash function to map a key value (or values) to a specific bucket or slot in the file. The hash function determines where the record is stored, enabling fast direct access if the key value is known.

4. Multitable Clustering File Organization

-> Multitable clustering involves physically grouping together records from two or more tables that are often accessed together. The goal is to reduce the cost of join operations by ensuring that related records are stored close to each other on disk.

Remote Backup System

-> A Remote Backup System in DBMS ensures data availability and reliability by maintaining a backup copy of a database at a remote location. This system is crucial for disaster recovery, preventing data loss due to system crashes, hardware failures, or natural disasters.



-> The primary database processes transactions and periodically sends log records to the backup system over the network.

-> The backup system continuously updates itself with the logs.

-> If the primary database crashes or becomes unavailable, the backup system can take over.

-> The backup database applies the latest log records to maintain consistency.

-> When the primary system fails, clients are redirected to the backup database, ensuring minimal downtime.

Advantages of Remote Backup Systems:

-> **High Availability**: Ensures that database services are available even if the primary server fails.
-> **Disaster Recovery**: Protects against data loss due to failures, crashes, or disasters.
-> **Minimal Data Loss**
-> **Automatic Failover**: Some systems allow automatic switching between the primary and backup servers.

Hashing

Hashing is a technique used in databases to efficiently retrieve records using a key. It maps a search key to a specific location in a data structure, typically a hash table, reducing the need for sequential searches.

Basic Concept of Hashing

-> **Hash Function (h(k))**: Converts a key k into an address in the hash table.
-> **Hash Table (Buckets)**: A structure storing data records indexed by the hash function.
-> **Collision Handling**: When multiple keys map to the same address, techniques like chaining or open addressing are used.

Types of Hashing in DBMS

There are two types of hashing techniques in DBMS:

1. Static Hashing (size of hash table fixed)

-> In static hashing, the number of buckets or slots in the hash table remains fixed. This works well when the number of records is predictable but can lead to issues if data grows or shrinks significantly.

-> The hash function always maps keys to a fixed number of buckets.

-> If the table becomes full, collisions increase.

-> Requires a reorganization of the table when the data grows beyond capacity.

Advantages of Static Hashing

-> Simple to implement.

-> Efficient for small and fixed datasets.

Disadvantages of Static Hashing

-> Inefficient for large datasets.

-> Requires periodic reorganization when data size changes.

2. Dynamic Hashing

-> Dynamic hashing allows the size of the hash table to grow and shrink as needed, making it more suitable for databases where data changes frequently.

Types of Dynamic Hashing

a. Extendible Hashing

-> Extendible hashing dynamically adjusts the number of buckets based on the number of keys.

Pros

-> Efficient in handling large and dynamic datasets.

-> Avoids excessive collisions.

Cons

-> Requires additional memory for the directory.

-> Slightly complex to implement.

b. Linear Hashing

-> Linear hashing allows gradual expansion of the hash table without a directory.

Advantages of Linear Hashing

-> No need for a directory, reducing overhead.

-> Grows progressively, reducing rehashing overhead.

Disadvantages of Linear Hashing

-> May require more time to retrieve records in some cases.

-> Can have uneven bucket distribution.

Indexing

-> Indexing in Database Management Systems (DBMS) is a technique used to improve the speed of data retrieval operations in a database. It creates an auxiliary data structure that allows the database to locate and fetch records faster without scanning the entire table.

-> works like index of the book to find required page no faster

Types of Indexing in DBMS

1. Ordered Indices (B-Tree, B+Tree)
2. Hash Indices (Static & Dynamic Hashing)

Ordered Indices	Hash Indices
1. The index entries are stored in a sorted order based on the indexed column(s).	a hash function is used to map values to fixed-size slots (buckets)
2. Typically implemented as a tree (e.g., B-tree or B+ tree)	Uses a hash table with buckets to store key-value pairs.
3. Efficient for range queries	Inefficient or not supported for range queries
4. Slower than hash indices	Extremely fast for exact matches
5. Best for range queries and sorting	Best for exact match searches
6. Used in traditional indexing (e.g., MySQL, PostgreSQL)	Used in NoSQL and caching systems (e.g., Redis, MongoDB)
7. Scales well	Can degrade with high collisions
8. Sequential Access Possible	Not Possible

Ordered Indices Types

1. Dense Index
2. Sparse Index

Dense Index	Sparse Index
1. Contains an index entry for every record in the database.	Contains an index entry for some records
2. Larger in size due to more index entries.	Smaller in size as it indexes fewer records.
3. Faster search, as every record is indexed.	Slower search, as it requires scanning blocks after finding the nearest index entry.
4. Requires more storage space	More space-efficient
5. Better for quick retrieval but less efficient for large datasets due to storage overhead.	Better for storage in large datasets but slower for retrieval.
6. Higher Update Overhead	Lower

B+ Tree Index in DBMS

A B+ Tree is a balanced tree data structure used for indexing in databases. It is an extension of the B-Tree and is commonly used to improve search efficiency for large datasets in Database Management Systems (DBMS) and file systems.

Why Use B+ Tree Indexing?

-> Efficient for range queries and sorted data retrieval.

-> Reduces disk I/O by keeping the index balanced.

-> Ensures logarithmic time complexity (O(log n)) for search, insert, and delete.

-> Leaf nodes store actual data for quick retrieval.

Structure of B+ Tree

-> A B+ Tree consists of internal nodes (index nodes) and leaf nodes (data nodes).

1. Internal Nodes (Index Nodes)

-> Store keys and pointers to child nodes.

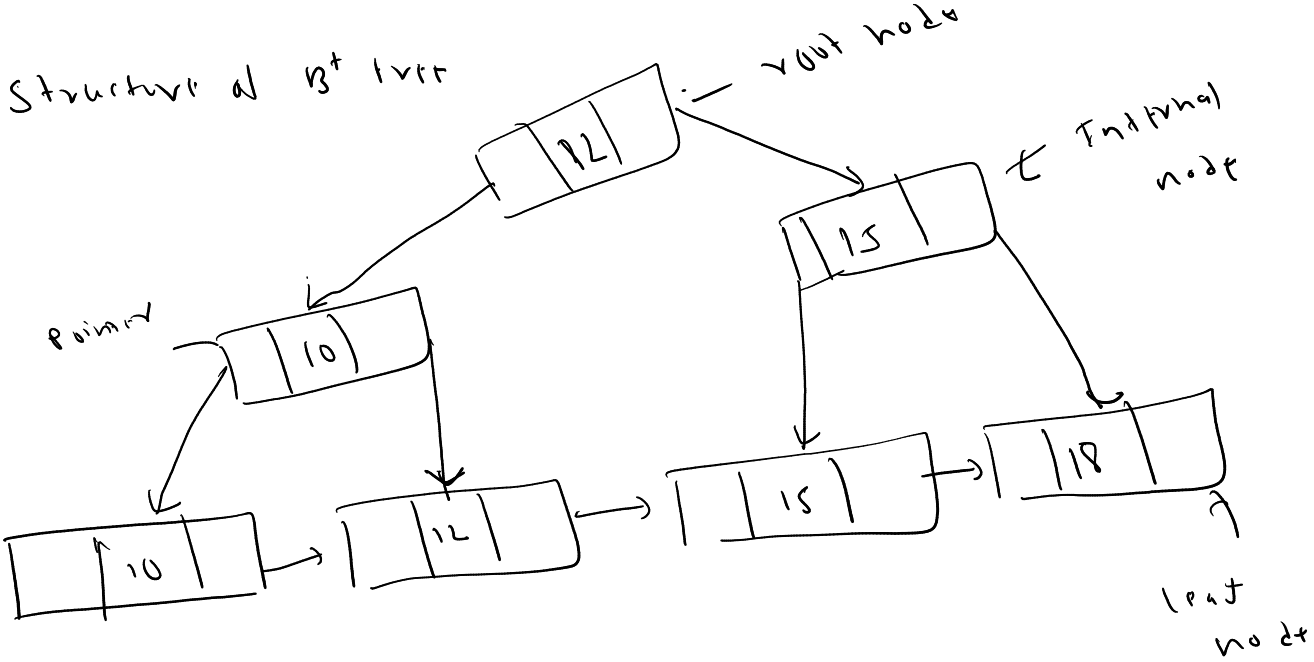
-> Do not store actual data.

-> Used for search navigation.

2. Leaf Nodes (Data Nodes)

-> Store actual data records (or pointers to them).

-> Are linked together to enable fast range queries.



Use Cases of B+ Tree Index

-> Databases (MySQL, PostgreSQL, Oracle) for primary and secondary indexing.

-> File Systems (NTFS, EXT4, HFS+) for efficient storage lookups.

-> Large-scale applications requiring fast range queries.

Advantages of B+ Tree

-> Efficient Searching (O(log n))

-> Supports Range Queries Efficiently

-> Less Disk I/O (Optimized for Databases)

-> Fast Insertion & Deletion (Auto-Balancing)

Disadvantages of B+ Tree

-> More Pointer Storage Overhead

-> Increased Complexity

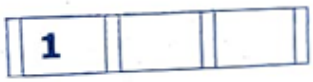
-> Not Ideal for Exact Lookups

Example in youtube video

B+ tree example by Kajchan Bhalp

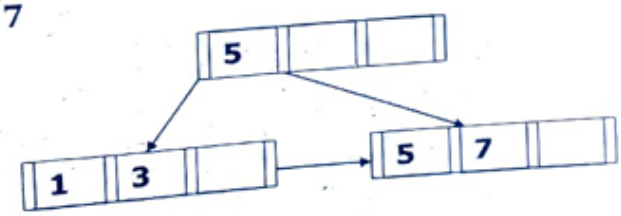
2. Perform the insertion operation in B+ tree for the following sequence
1, 3, 5, 7, 9, 2, 4, 6, 8, 10 [2079 Jestha]

Ans: Insert 1

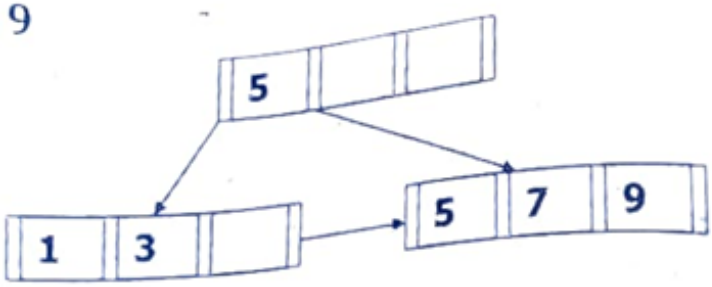


Insert 3,5

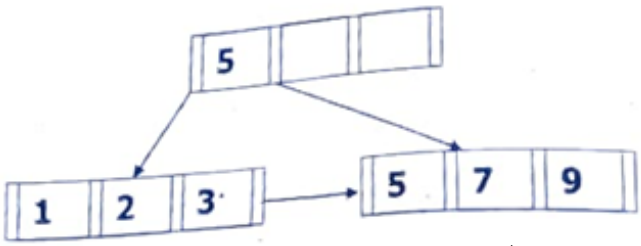
Insert 7



Insert 9

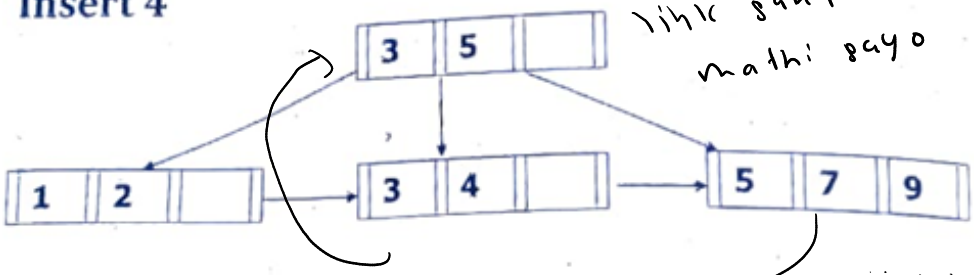


Insert 2



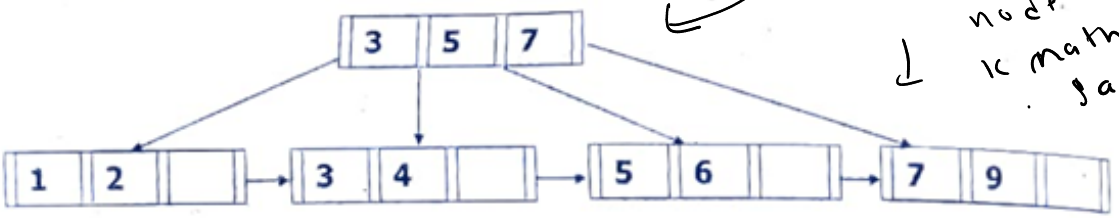
Insert 4

moved to another node so like yaha mathi gayo

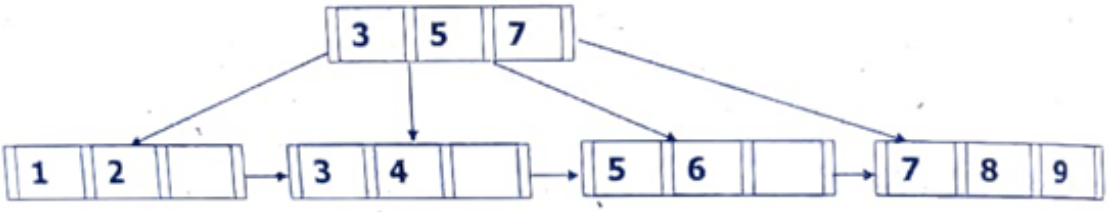


Inset 6

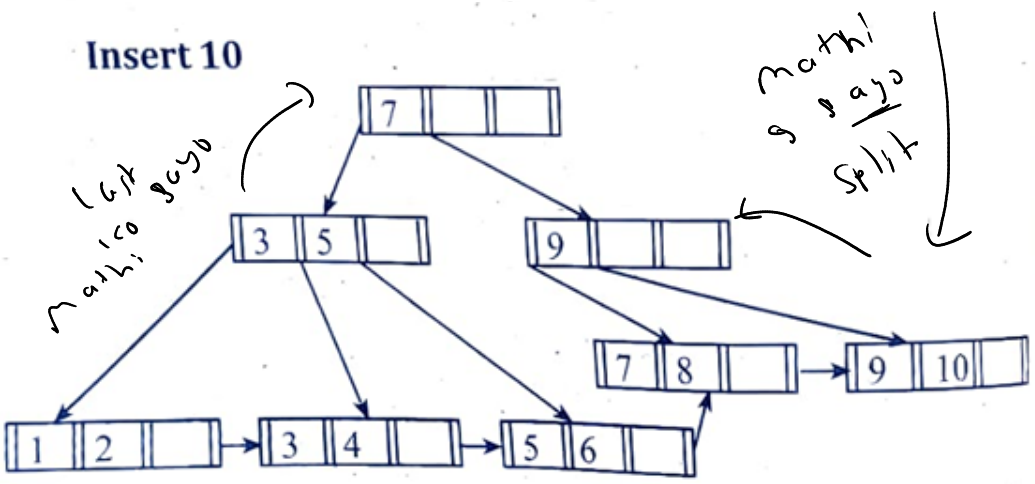
so like this node ko mathi gayo



Insert 8



Insert 10



4. What factors to be taken into account in choosing a RAID level? [2078 Bhadra]

-> Redundancy & Fault Tolerance
-> Performance
-> Storage Efficiency
-> Failure Tolerance
-> Minimum Disk Requirement
-> Cost
-> Best for Specific Use Cases
-> Data Availability
-> Capacity
-> Compatibility