Great Place To Work® Certified
MAR 2022 - MAR 2023
INDIA

UPES

UNIVERSITY OF TOMORROW

nirf Ranking 2022

Ranked 65th University in India

Accredited with Grade A NAAC

Accredited **Grade** 'A' by NAAC

QS STARS™ RATING SYSTEM 2020

QS 5 Star Rating for Academic Development, Employability, Facilities and Program Strength

E-LEAD

Perfect score of **150/150** as a testament to exceptional E-Learning methods

Saroj Shivagunde

Unit 1

Disk Storage, File Structures, and Indexing

Saroj Shivagunde

Overview

- Introduction
- Secondary Storage Devices
- Buffering of Blocks and Placing File Records on Disk
- File Operations
- Heap Files, Sorted Files
- Hashing Techniques
- Parallelizing Disk Access using RAID Technology
- Indexing

Introduction

Introduction

- This unit covers storage and retrieval part of the DBMS
- We dive below the higher levels of DBMS to study underlying structures to take a look at-
  - Various methods and structures used by Database Management Systems (DBMS) to store, manage, and retrieve data efficiently
  - Buffering strategies for temporarily storing data in memory to enhance performance and reduce disk I/O operations
  - The use of different file types to optimize data access and manipulation
  - Transforming data into a fixed-size value (hash code) for efficient data retrieval by mapping keys to specific locations in a hash table
  - Use of RAID technology to improve data redundancy, fault tolerance, and performance, with various RAID levels offering different balances of these benefits
  - Use of indexing for quick access to records based on key values

Secondary Storage Devices

Physical Storage Media : Primary Memory

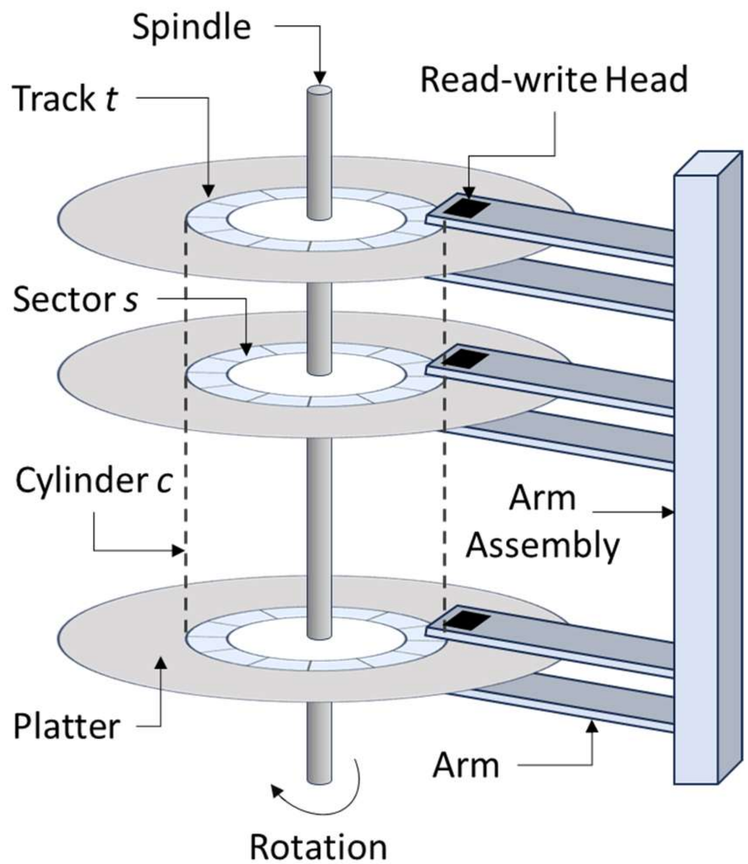
- Primary memory is volatile memory used to store data temporarily while a computer is running
- Cache
  - A small, fast memory component that stores frequently accessed data
  - Much faster than main memory (RAM)
  - Often used in CPUs to store critical instructions and data
- Main Memory (RAM)
  - Primary volatile memory to store data and programs that are in use
  - Faster than secondary storage but slower than cache
  - Data is lost when the power is turned off
  - Used for running applications and the operating system

Physical Storage Media : Secondary Memory

- Secondary memory is non-volatile memory used to store data for a long-term, even when the computer is powered off
- Optical Storage
  - Uses laser technology to read and write data on optical discs such as CDs, DVDs, and Blu-ray discs
  - Good for long-term storage and resistant to magnetic interference
  - Capacity varies by type (CDs: up to 700 MB, DVDs: up to 4.7 GB, Blu-ray: up to 50 GB)
  - Used for media distribution, backups, and archival purposes.
- Flash Memory
  - Non-volatile memory for storage and transfer of data (SSDs, USB drives, and memory cards)
  - Retains data without power
  - Faster read and write speeds compared to magnetic disks

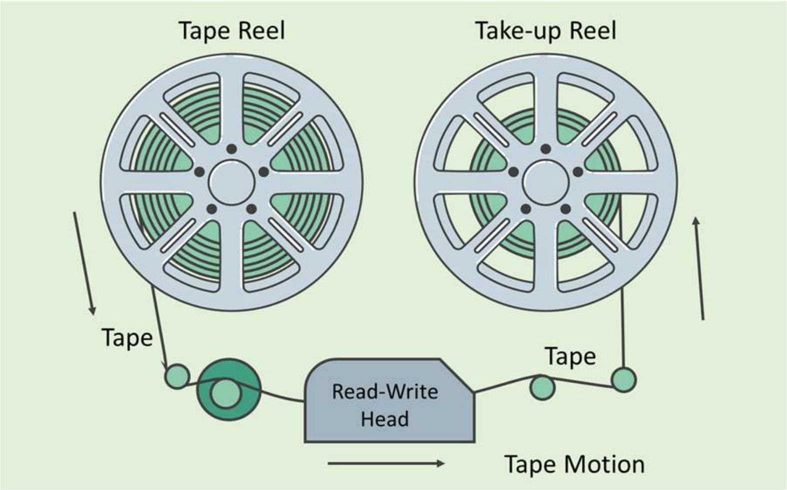
Other Secondary Storage Devices : Magnetic Disks

- Non-volatile storage media that uses magnetic patterns to store data
- Offers large storage capacities at a lower cost
- Slower read and write speeds compared to flash memory
- Commonly used in traditional hard drives (HDDs) and enterprise storage systems
- The primary medium for the long-term online storage of data
- Provides a bulk of secondary storage for modern computer systems



Other Secondary Storage Devices : Magnetic Tapes

- Used for data storage, primarily for backup and archival purposes
- High storage capacity, suitable for large data volumes
- Cost-effective for long-term storage
- Slower access times compared to disk-based storage
  - as the access is sequential rather than random



- Tape libraries are used to hold exceptionally large collections of data as much as hundreds of terabytes or even multiple petabytes
  - These libraries are called jukeboxes

Exercise

- **Practice Questions:**
  - List and describe three types of secondary storage devices.
  - Explain the impact of latency on database performance.
  - How do we calculate throughput of a secondary storage device?

- **Interview Questions:**
  - What are the key differences between HDDs and SSDs?

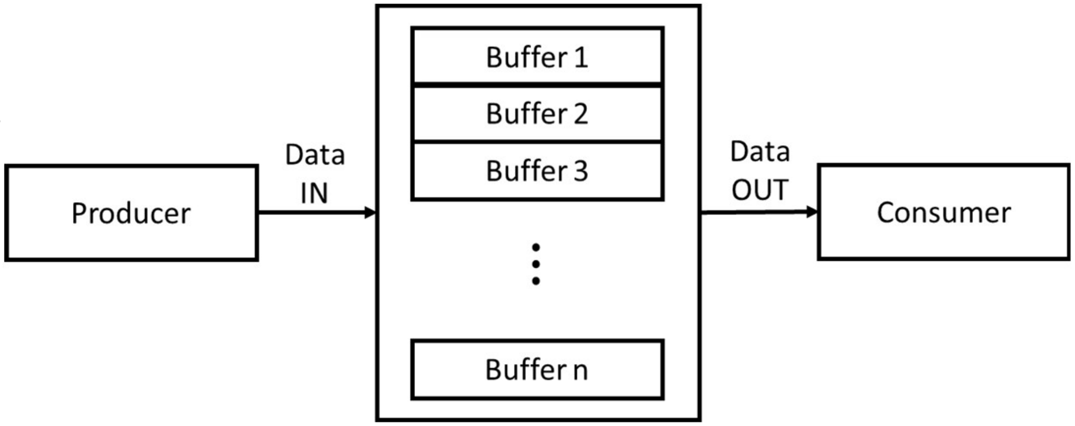
Buffering of Blocks and Placing File Records on Disk

Buffering of Blocks

- A buffer is the temporary storage of data while it is being moved from one place to another
- Buffering facilitates:
  - **Smooth Data Transfer**: by accommodating differences in data processing speeds between the source and destination
  - **Reduced Latency**: by allowing data to be sent or received in larger, more manageable chunks
  - **Error Handling**: by providing a space to store data temporarily while the system resolves issues.
  - **Increased Throughput**: By using buffers, systems can increase throughput as data can be processed in larger batches, reducing the overhead associated with frequent data transfers
  - Resource Management: Buffers help in better resource management by allowing asynchronous processing, where the producer and consumer of data can operate at different speeds without blocking each other.

Buffering of Blocks : Types of Buffering

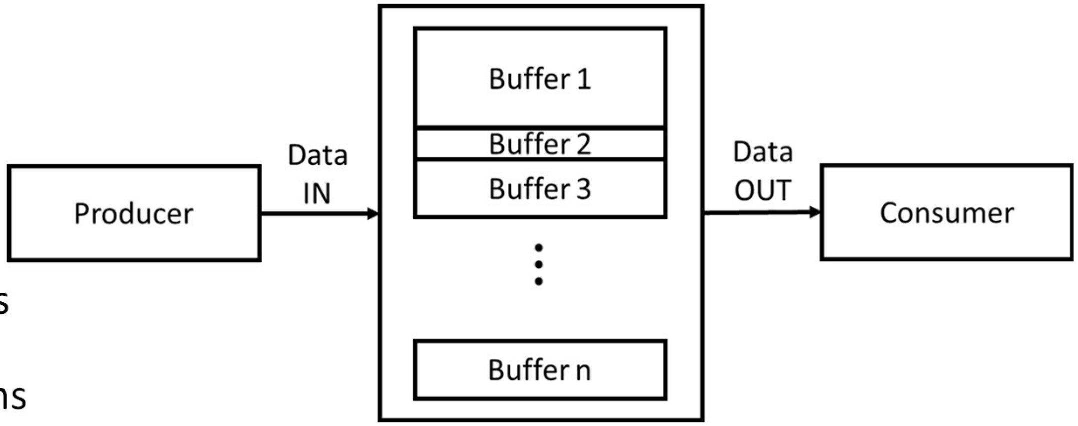
- Fixed-size Buffering
  - The buffer has a predetermined, unchangeable size
- Used in systems where the amount of data is known and consistent
  - DBMSs use fixed-size buffering for managing B-Tree indexes
  - Transaction logs in databases are managed using fixed-size buffering



- It is simple in implementation and management
- If the buffer size is not optimally chosen, it may lead to wasted memory or buffer overflow

Buffering of Blocks : Types of Buffering

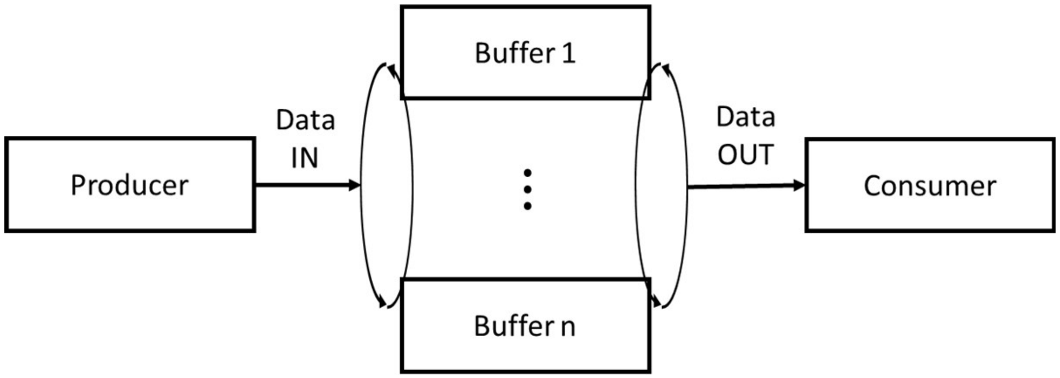
- Dynamic Buffering
  - Allows the buffer size to be adjusted based on the amount of data
- Used in scenarios where the data volume is unpredictable
  - DBMS uses dynamic buffering for queries that may require variable sized buffers such as- join operations or sort operations



- Efficient memory usage as the buffer can expand or contract as needed
- More complex to implement and manage, potentially leading to overhead in adjusting buffer sizes dynamically

Buffering of Blocks : Types of Buffering

- Circular Buffering
  - Uses a buffer that wraps around to the beginning when it reaches the end
- Commonly used in streaming data applications
  - Such as audio or video playback, where continuous data flow is required



- Efficient use of memory as old data is overwritten by new data, avoiding the need for buffer resizing.
- Complexity in managing the buffer pointers and ensuring data consistency, especially in multi-threaded environments.

Methods for Placing File Records on Disk

- Contiguous Allocation
  - All records of a file are stored in sequential disk blocks
  - **Advantages**- fast access and simplicity
  - **Disadvantages**- may lead to fragmentation and difficulties in file expansion
- Linked Allocation
  - Each file is a linked list of disk blocks, and each block contains a pointer to the next block
  - **Advantages**- eliminates fragmentation
  - **Disadvantages**- may cause slower access times due to scattered blocks
- Indexed Allocation
  - Uses an index block containing pointers to all the file's blocks
  - **Advantages**- efficient direct access to any block, simplified file growth
  - **Disadvantages**- needs additional space for the index

Methods for Placing File Records on Disk

- Clustered Allocation
  - Groups multiple contiguous blocks together in clusters
  - Combines the benefits of contiguous allocation and linked allocation
  - **Advantages**- simplicity and reduced fragmentation
  - **Disadvantages**- better suited to files that grow in predictable patterns
- Hashed Allocation
  - Applies a hash function to determine the disk block address for a record
  - Ensures uniform distribution of records and minimizes collision handling
  - useful for random access patterns

Exercise

- **Practice Questions:**
  - What is block buffering?
  - Describe two methods for placing file records on disk.
  - Differentiate between Contiguous Allocation and Linked Allocation.

- **Interview Questions:**
  - How does block buffering improve database performance?

File Operations

File Operations

- Fundamental actions performed on files to facilitate effective data management and manipulation
- Create (C)
  - Generating a new file within the filesystem, assigning it a unique name, and allocating the necessary disk space for storing data
- Read (R)
  - Retrieves data from a file, allowing access and use of the stored information without modifying it
- Update (U)
  - Modifying its existing data by altering, appending, or overwriting the content of the file
- Delete (D)
  - Removes a file from the filesystem, frees up the allocated disk space and makes the file name available for reuse

File Access Methods : Sequential Access

- Sequential access reads or writes data records in a linear order
  - One after the other, from the beginning to the end of the file
- Use Cases
  - Ideal for applications where data is processed in a predictable linear fashion
  - For e.g.- log files, batch processing, and media streaming
- Advantages
  - Simple and efficient for accessing large volumes of data sequentially
  - Minimizes seek time and improves throughput
- Limitations
  - Inefficient for random access patterns, as accessing data near the end of the file requires traversing all preceding records.

File Access Methods : Random Access

- Random access allows data records to be read or written in any order
  - Direct access of the desired location within the file without sequential traversal
- Use Cases
  - Suitable for applications requiring quick access to specific records
  - For e.g.- databases, indexing, and certain types of real-time data processing
- Advantages
  - Provides flexibility and speed for accessing individual records
  - Reduces latency for non-sequential data retrieval operations
- Limitations
  - May involve higher complexity in implementation
  - Increased overhead due to potential frequent seek operations and data fragmentation

Exercise

- **Practice Questions:**
  - What are CRUD operations?
  - Explain the difference between sequential and random file access.

- **Interview Questions:**
  - Describe a scenario where random file access is preferable over sequential access.

Heap Files, Sorted Files

Heap Files

- Unordered collections of records stored in a file
  - where new records are appended to the end of the file, without any specific order
- Use Cases
  - Small or simple databases, temporary storage
  - Scenarios where fast insertion is prioritized over data retrieval speed
- Advantages
  - Efficient for insertion operations as new records are added directly to the end of the file
  - No reorganization or indexing is required
- Disadvantages
  - Slower retrieval compared to ordered files because a linear search may be necessary to locate a specific record
  - May lead to fragmentation over time as records are added and deleted, potentially requiring periodic compaction or reorganization to optimize space

Sorted Files

- Store records in a specific order based on one or more key fields
  - ensures that records are maintained in a sorted sequence
- Use Cases
  - Ideal for applications requiring efficient search operations, range queries, and ordered data retrieval, such as directory services, and report generation
- Advantages
  - Provides faster search capabilities compared to heap files due to the ordered structure
  - Enhances read performance for sequential access and range queries
- Disadvantages
  - File operations can be more complex and time-consuming as they require maintaining the sorted order
  - May incur higher overhead for write operations
  - Regular maintenance or reorganization may be necessary to optimize performance, especially after frequent insertions and deletions.

Sorting Algorithms Used in File Organization

- Merge Sort
  - A divide-and-conquer algorithm that divides the file into smaller subfiles, sorts them, and then merges the sorted subfiles back together
  - Ideal for large datasets stored on disk
  - Requires memory in orders of O(n)
- Quick Sort
  - Another divide-and-conquer algorithm that selects a 'pivot' element and partitions the records into two subsets, sorting them recursively
  - Efficient but can be less stable and require more memory
- Heap Sort
  - Uses a binary heap data structure to sort records, offering a good balance of time complexity and space efficiency
  - Suitable for in-memory sorting before writing to disk

Sorting Algorithms Used in File Organization

- Insertion Sort
  - Simple algorithm that builds the sorted array one item at a time, placing each new element in its correct position
  - Practical for small files or partially sorted data
- External Sort
  - Involves reading data into memory in chunks, sorting each chunk, and then merging the sorted chunks
  - Designed for handling large files that do not fit into main memory
- Bubble Sort
  - A simple comparison-based algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order
  - Generally inefficient for large datasets but educationally useful

Exercise

- **Practice Questions:**
  - List advantages and disadvantages of heap files.
  - Describe a scenario where sorted files are advantageous than heap files.
  - Given a large dataset in a DBMS that must be sorted but cannot be entirely loaded into memory at once, which sorting algorithm would you choose and why?

- **Interview Questions:**
  - When would you choose a heap file structure in a database application?
  - How do sorted files improve query performance?
  - Compare the time complexities and practical considerations of using merge sort, quick sort, heap sort, and bubble sort for sorting data within a database.
  - How do you decide between using a heap file and a sorted file?

Hashing Techniques

Hash Functions And Hash Tables

- Hash functions map input data (keys) to a fixed-size string of bytes (hash code)
  - which is used to index a hash table where the actual values are stored
- Hash tables provide efficient data retrieval
  - Average-case time complexity of O(1) for search, insert, and delete operations
  - Ideal for applications requiring fast lookups.
- Advantages
  - **Deterministic Nature**: A given input key will always produce the same hash code, ensuring consistency in data storage and retrieval
  - **Uniform Distribution**: A good hash function distributes keys uniformly across the hash table to minimize collisions and ensure efficient utilization of the table space.
- Limitations
  - A good hash function and table size must be chosen to minimize collisions and optimize performance
  - Load factors and rehashing strategies must be kept in mind too

Collision Resolution Methods

- Chaining
  - Each hash table index points to a linked list of entries that share the same hash code
  - Multiple entries can exist at the same index without overwriting
- Linear Probing
  - Sequentially checks the next available slots in the hash table to find an empty one
  - Simple but may lead to clustering issues
- Quadratic Probing
  - Uses a quadratic function to find the next available slot
  - Reduces clustering compared to linear probing but complicating the probing sequence

Collision Resolution Methods

- Double Hashing
  - A collision resolution technique used in open addressing within hash tables
  - Applies a second hash function to calculate the probe sequence when collision occurs while using first hash function
  - Providing a more uniform distribution of entries and reduces clustering
- Open Addressing
  - Resolves collisions by finding another open slot within the hash table through probing methods
  - Methods used for probing- linear probing, quadratic probing, or double hashing
- Rehashing
  - Helps to maintain performance and reduce collisions when the hash table becomes too full
  - A new larger table is created, and all entries are reinserted using the original hash function

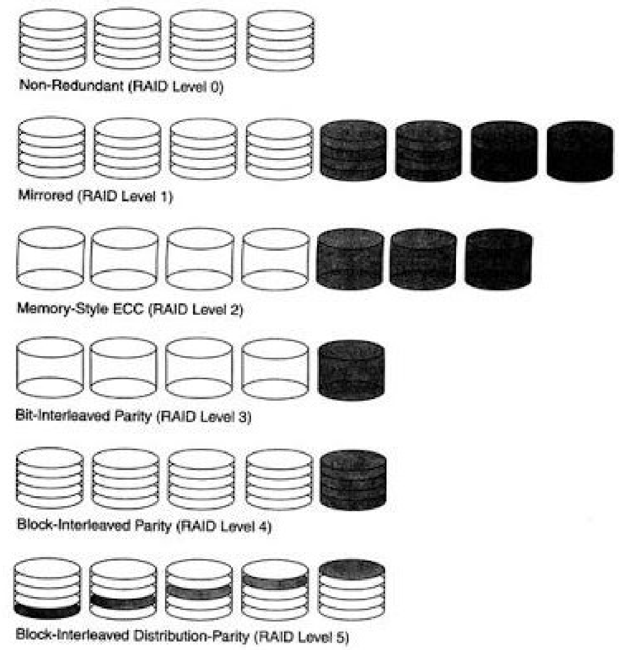
Exercise

- **Practice Questions:**
  - Explain the purpose of a hash function.
  - List and describe two collision resolution methods.
  - Describe the process of creating a hash index.
- **Interview Questions:**
  - How is hashing used in database indexing?
  - How does rehashing help if the same hash function is used for the new table too?
  - How does double hashing avoid collisions?

Parallelizing Disk Access using RAID Technology

RAID (Redundant Array of Independent Disks)

- RAID (Redundant Array of Independent Disks)
  - a data storage virtualization technology that combines multiple physical disk drives into a single logical unit to improve performance and data redundancy
- RAID Levels
  - Different RAID levels offer different strategies to meet specific needs for speed, redundancy, and fault tolerance
- RAID is widely used in enterprise environments, data centers, and servers
  - It ensures data availability, reliability, and improved storage performance, tailored to specific application requirements



RAID levels

- RAID Level 0
  - Data is striped across multiple disks without redundancy
  - **Advantage**: Provides high performance with increased read and write speeds
  - **Disadvantage**: Offers no data protection. A single disk failure results in complete data loss
- RAID Level 1
  - Data is mirrored across two disks
  - **Advantage**: Ensures data redundancy and high availability, as each disk is an exact copy
  - **Disadvantage**: Storage efficiency is low, as only half the total disk space is usable
- RAID Level 2
  - Data is striped at the bit level with error correction (Hamming code)
  - **Advantage**: Provides error detection and correction with high data integrity
  - **Disadvantage**: Complex implementation and rarely used due to the high cost and inefficiency with modern disks

RAID levels

- RAID Level 3
  - Data is striped at the byte level with a dedicated parity disk
  - **Advantage**: Offers good read and write performance with single-disk failure protection
  - **Disadvantage**: Parity disk can become a bottleneck. Better alternatives are available
- RAID Level 4
  - Data is striped at the block level with a dedicated parity disk
  - **Advantage**: Allows for efficient large block data transfers with single-disk failure protection
  - **Disadvantage**: Parity disk can become a bottleneck, impacting write performance
- RAID Level 5
  - Data and parity are striped across all disks
  - **Advantage**: Balances good performance, storage efficiency, and data protection with single-disk failure tolerance
  - **Disadvantage**: Write operations can be slower due to parity calculation and distribution

Exercise

- **Practice Questions:**
  - What is RAID?
  - Why are there different RAID levels?
  - Describe the differences between RAID 0, 1, and 5.

- **Interview Questions:**
  - What are the advantages of using RAID 5?

Indexing

Indexing : Primary Index

- Indexes are auxiliary data structures that provide quick access to records based on key values
- Primary Index
  - Directly related to the primary key of a table and is automatically created when the primary key is defined, typically stored in a sorted order
  - Ensures that the index entries are unique, corresponding to each unique primary key value
  - Single-level ordered indexes are same as primary indexes, only they use other key attributes
- Benefits
  - Can quickly locate records based on the primary key, providing efficient and fast access
  - Sorted and unique values facilitate fast binary searches and range queries on the primary key
  - Automatically maintains data integrity and consistency as it is based on a primary key
- Limitations
  - Any changes (updates/deletions) to the primary key affect it as it is tied to the primary key
  - No benefit for queries involving non-primary key attributes or complex conditions

Indexing : Secondary Index or Secondary Access Paths

- Secondary Indexes or Secondary Access Paths
  - They provide alternative methods for accessing and retrieving data from a database
  - Commonly used in scenarios where frequent read operations on specific non-primary key columns are required
  - Various types of secondary indexes, including B-trees, B+-trees, hash indexes, and bitmap indexes, can be employed depending on the specific access patterns and query requirements
- Benefits
  - They are independent of the primary key or physical storage order
  - They improve the efficiency of query operations by allowing faster data retrieval for non-primary key attributes, significantly reducing search times
  - They are crucial for supporting complex query operations such as those involving range searches, joins, and filtering on non-key attributes
- Limitations
  - They introduce some level of data redundancy and require additional storage space

Types of Indexes

- Clustered Index
  - The table's data rows are stored in the order of the index key, defining the physical order of data in the table
  - **Advantages**: Provides efficient range queries and sequential access due to the sorted order of data
  - **Disadvantages**: Only one clustered index can be created per table because the data rows can be stored in only one order
- Non-Clustered Index
  - The index contains pointers to the actual data rows rather than defining their physical order
  - **Advantages**: Multiple non-clustered indexes can be created on a table, allowing for flexible querying on various columns
  - **Disadvantages**: Can result in additional I/O operations to fetch the actual data rows, as the index and data are stored separately

Types of Indexes

- Unique Index
  - Ensures that the indexed columns do not have duplicate values, maintaining data uniqueness and integrity
  - **Advantages**: Enhances query performance for unique key lookups and enforces data integrity by preventing duplicates
  - **Disadvantages**: Overhead in maintaining uniqueness during insertions and updates, potentially impacting performance
- Composite Index
  - Created on multiple columns of a table, which can be either clustered or non-clustered
  - **Advantages**: Improves performance for queries that filter, or sort based on multiple columns, as the combined columns are indexed
  - **Disadvantages**: Can be larger and more complex to maintain, potentially affecting insert, update, and delete operations due to the multi-column structure

Types of Indexes

- Dense Index
  - Has an entry for every record in the table, maps each key value to the corresponding disk block
  - **Advantage**: Provides fast access to any record since every search key value is indexed
  - **Disadvantage**: Requires more storage space and maintenance overhead than sparse indexes
- Sparse Index
  - Has entries for only some of the records, typically one entry per disk block
  - **Advantage**: Requires less storage space and maintenance effort
  - **Disadvantage**: Slower access times than dense indexes as it requires additional block accesses
- Multi-Level Index
  - Indexes are structured in multiple levels to handle large datasets efficiently, with the top-level index pointing to second-level indexes, and so on
  - **Advantage**: Improved search efficiency due to reduced disk accesses needed to find a record
  - **Disadvantage**: Complex to maintain the multi-level structure, especially during insertions and deletions

Types of Indexes

- B-Tree Index
  - A balanced tree structure where all leaf nodes are at the same level
  - Internal (non-leaf) nodes store keys and pointers to child nodes, and data records are stored in both internal and leaf nodes
  - Leaf nodes can contain both keys and data pointers, allowing data to be retrieved from any node
  - **Advantage**: Logarithmic search time, efficient insertion, deletion, and search operations
  - **Disadvantage**: Complex to implement and maintain than simpler index structures
- B+ Tree Index
  - An extension of B-trees, where all values are stored at the leaf level, and internal nodes only store keys for guiding the search
  - All leaf nodes are linked sequentially, making in-order traversal straightforward and efficient
  - **Advantage**: Provides efficient range queries and sequential access, ideal for database indexing
  - **Disadvantage**: May require more space for storing redundant copies of keys in internal nodes

Exercise

- **Practice Questions:**
  - Explain how secondary indexes improve query performance.
  - What is a single-level ordered index?
  - Compare dense and sparse indexes.
  - Describe the structure of a multilevel index.
  - Differentiate between clustered and non-clustered primary index.
  - What does a primary index ensure in a database table?
- **Interview Questions:**
  - When should a secondary index be used?
  - Explain a scenario where a multilevel index is essential for performance.
  - How does a B+ Tree differ from a B-Tree?
  - How single-level ordered index is different than a primary index?
  - Explain the benefits of using a composite primary index.

Thank You

Saroj Shivagunde