UNIT-4

FILE ORGANIZATION

-The database is stored as a collection of **files**.
- Each file is a sequence of **records**.
-A **record** is a sequence of fields.
-Classifications of records

i. **Fixed length record**
ii. **Variable length record**

Fixed length record approach:

b. assume record size is fixed
c. each file has records of one particular type only
d. Different files are used for different relations. This case is easiest to implement.

- Simple approach:
  - Record access is simple
    - Modification: do not allow records to cross block boundaries

Variable length record

- Deletion of record I:
  alternatives:
  - move records i + 1, . . ., n to i, . . . , n – 1
  - do not move records, but link all free records on a free list
- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.

- Byte string representation
  - Attach an end-of-record (⊥) control character to the end of each record
  - Difficulty with deletion

Slotted Page Structure

Slotted page header contains:

- number of record entries
- end of free space in the block
- location and size of each record

Free Lists

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on

Pointer method

- A variable-length record is represented by a list of fixed-length records, chained together via pointers.
- Can be used even if the maximum record length is not known

RAID TECHNIQUES

RAID: Redundant Arrays of Independent Disks

a. disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
   i. high capacity and high speed by using multiple disks in parallel, and

ii. high reliability by storing data redundantly, so that data can be recovered even if a disk fails

RAID Level 0: striping; non-redundant.

- Used in high-performance applications where data lost is not critical.

RAID Level 1: **Mirroring** (or **shadowing**)

- Duplicate every disk.
- Every write is carried out on both disks
- If one disk in a pair fails, data still available in the other
  - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
    - Probability of combined event is very small

- **RAID Level 2**: Error-Correcting-Codes (ECC) with bit striping.
- **RAID Level 3**: Bit-Interleaved Parity
  - a single parity bit is enough for error correction, not just detection, since we know which disk has failed
    - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
    - To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
- **RAID Level 4:** Block-Interleaved Parity.
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.
- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all N + 1 disks, rather than storing data in N disks and parity in 1 disk.
- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.
  -

SECONDARY STORAGE DEVICES

Can differentiate storage into:

b. **volatile storage:** loses contents when power is switched off
c. **non-volatile storage**:
   i. Contents persist even when power is switched off.
   ii. Includes secondary and tertiary storage.

Primary storage: Fastest media but volatile (cache, main memory).

Secondary storage: next level in hierarchy, non-volatile, moderately fast access time also called **on-line storage**

Tertiary storage: lowest level in hierarchy, non-volatile, slow access time also called **off-line storage**

E.g. magnetic tape, optical storage

Cache – fastest and most costly form of storage; volatile; managed by the computer system hardware.

Main memory:

- fast access
- generally too small
- capacities of up to a few Gigabytes widely used currently

- **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.

Flash memory

- Data survives power failure
- Data can be written at any location.
- The location can be erased and written to again
- Can support only a limited number of write/erase cycles.
- Reads are fast.
- But writes are slow (few microseconds), erase is slower
- also known as EEPROM (Electrically Erasable Programmable Read-Only Memory)

Magnetic-disk

a. Data is stored on spinning disk, and read/written magnetically
b. Primary medium for the long-term storage of data.
c. Data must be moved from disk to main memory for access, and written back for storage
    i. Much slower access than main memory
d. **direct-access** – possible to read data on disk in any order, unlike magnetic tape
    i. Much larger capacity than main memory/flash memory
    ii. disk failure can destroy data, but is very rare

Read-write head

-Positioned very close to the platter surface
-Reads or writes magnetically.

-Surface of platter divided into circular **tracks**
-Each track is divided into **sectors.**
-A sector is the smallest unit of data that can be read or written.
-Head-disk assemblies
-multiple disk platters on a single spindle (typically 2 to 4)
-ne head per platter, mounted on a common arm.

Cylinder i consists of i^{th} track of all the platters

Disk controller – interfaces between the computer system and the disk drive hardware.

- accepts high-level commands to read or write a sector
- initiates actions such as moving the disk arm to the right track and actually reading or writing the data
- Ensures successful writing by reading back sector after writing it.

- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM and DVD most popular forms
  - Write-one, read-many (WORM) optical disks are available (CD-R and DVD-R)
  - Multiple write versions also available (CD-RW, DVD-RW)
  - Reads and writes are slower than with magnetic disk
- **Tape storage**
  - Non-volatile, Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another..
- Hold large volumes of data and provide high transfer rates
  - **sequential-access** – much slower than disk
- Very slow access time in comparison to magnetic disks and optical disks
  - very high capacity (300 GB tapes available)
  - storage costs much cheaper than disk, but drives are expensive

STATIC AND DYNAMIC HASHING

Static hashing:

A bucket is a unit of storage containing one or more records.

In a hash file organization we obtain the bucket of a record directly from its search-key value using a hash function.

Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B.

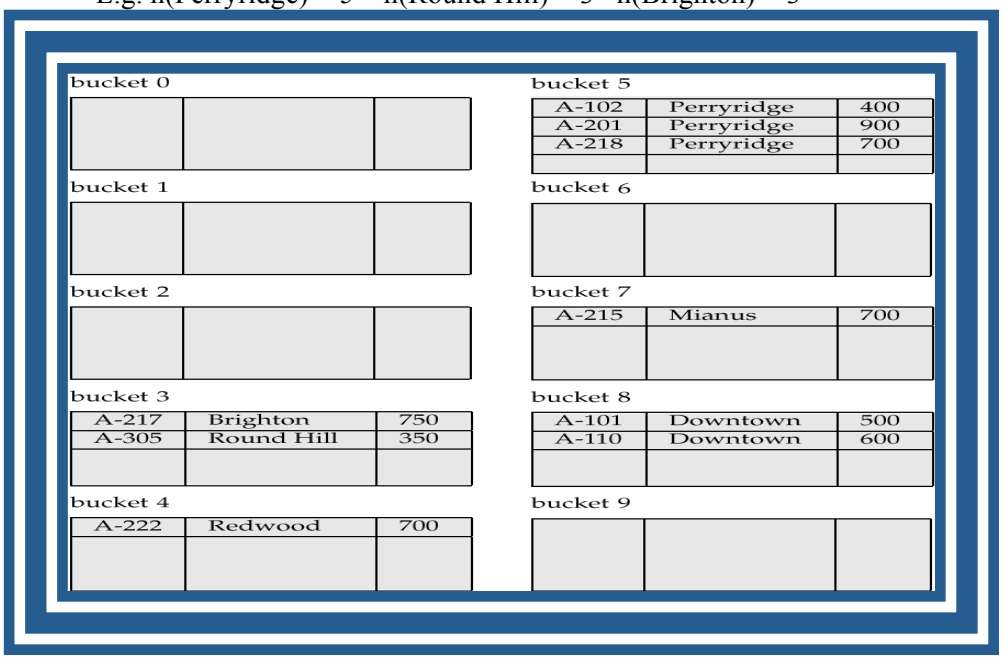
Hash function is used to locate records for access, insertion as well as deletion.

Example:

-There are 10 buckets,

-The hash function returns the sum of the binary representations of the characters modulo 10

– E.g. h(Perryridge) = 5 h(Round Hill) = 3 h(Brighton) = 3



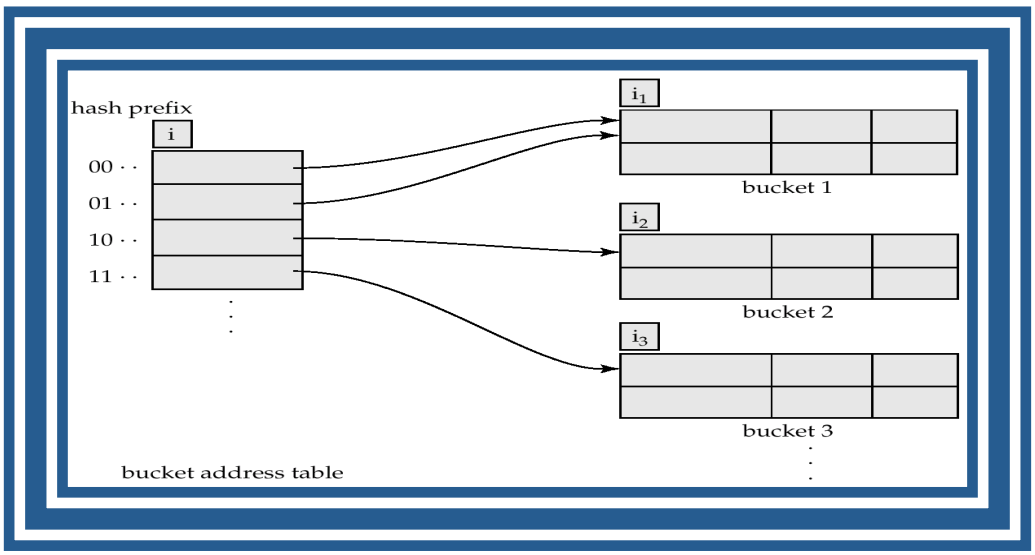
Dynamic hashing:

- Good for database that grows and shrinks in size
- Allows the hash function to be modified dynamically
- Extendable hashing – one form of dynamic hashing
  - Hash function generates values over a large range — typically b-bit integers, with

b = 32.

- At any time use only a prefix of the hash function to index into a table of bucket addresses.
- Let the length of the prefix be i bits, $0 \le i \le 32$.
- Bucket address table size = 2^i. Initially i = 0
- Value of i grows and shrinks as the size of the database grows and shrinks.
- Multiple entries in the bucket address table may point to a bucket.
- Thus, actual number of buckets is $< 2^i$
  - The number of buckets also changes dynamically due to coalescing and splitting of buckets.

General Extendable Hash Structure



In this structure, $i_2 = i_3 = i$, whereas $i_1 = i - 1$ (see next slide for details)

Use of Extendable Hash Structure

- To locate the bucket containing search-key K_j:
  1. Compute $h(K_j) = X$
  2. Use the first i high order bits of X as a displacement into bucket address table, and follow the pointer to appropriate bucket

Updates in Extendable Hash Structure

- To insert a record with search-key value K_j
  - Follow same procedure as look-up and locate the bucket, say j.
  - If there is room in the bucket j insert record in the bucket.
  - Overflow buckets used instead in some cases.
- To delete a key value,
  - Locate it in its bucket and remove it.
  - The bucket itself can be removed if it becomes empty
  - Coalescing of buckets can be done
  - Decreasing bucket address table size is also possible

ORDERED INDICES

In an **ordered index,** index entries are stored sorted on the search key value.

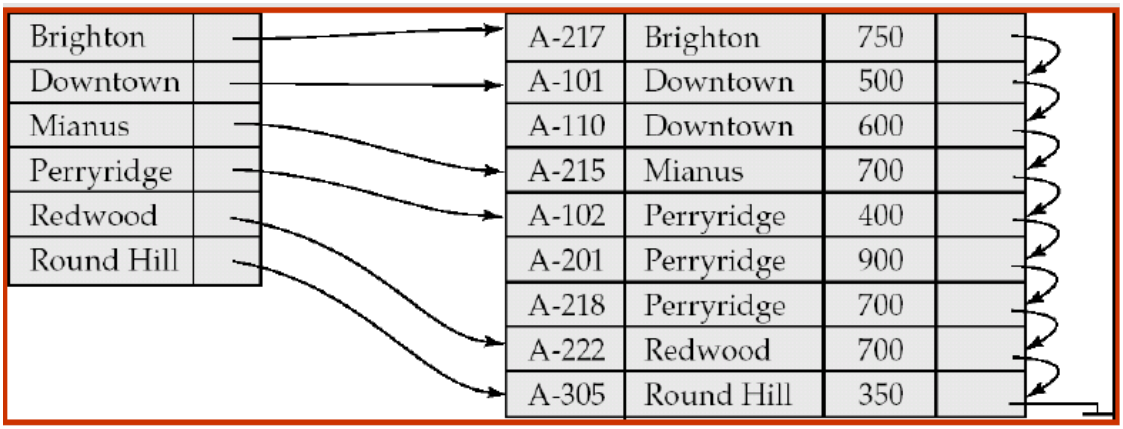
Primary index: in a sequentially ordered file, the index whose search key specifies the sequential order of the file.

Secondary index: an index whose search key specifies an order different from the sequential order of the file.

Types of Ordered Indices

- Dense index
- Sparse index

Dense index — Index record appears for every search-key value in the file.

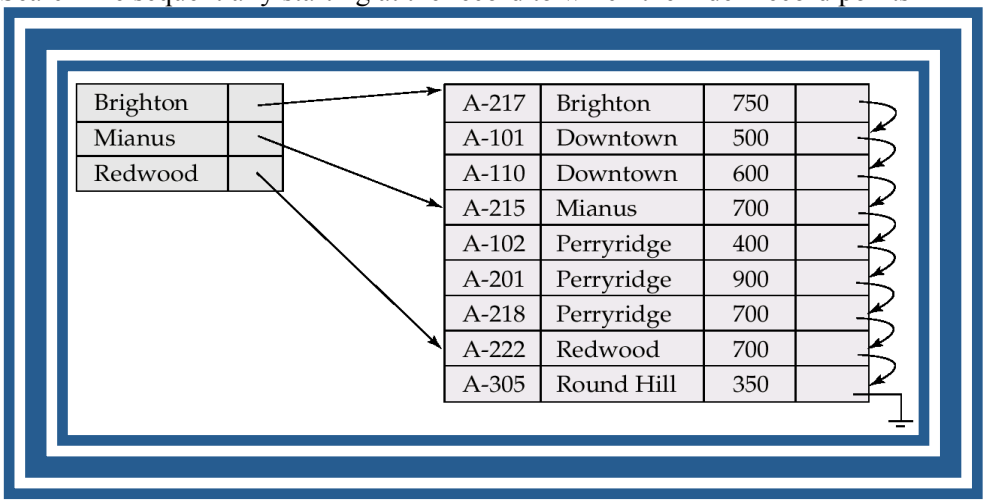


Sparse Index: contains index records for only some search-key values.

- Applicable when records are sequentially ordered on search-key

To locate a record with search-key value K we:

- Find index record with largest search-key value < K
- Search file sequentially starting at the record to which the index record points



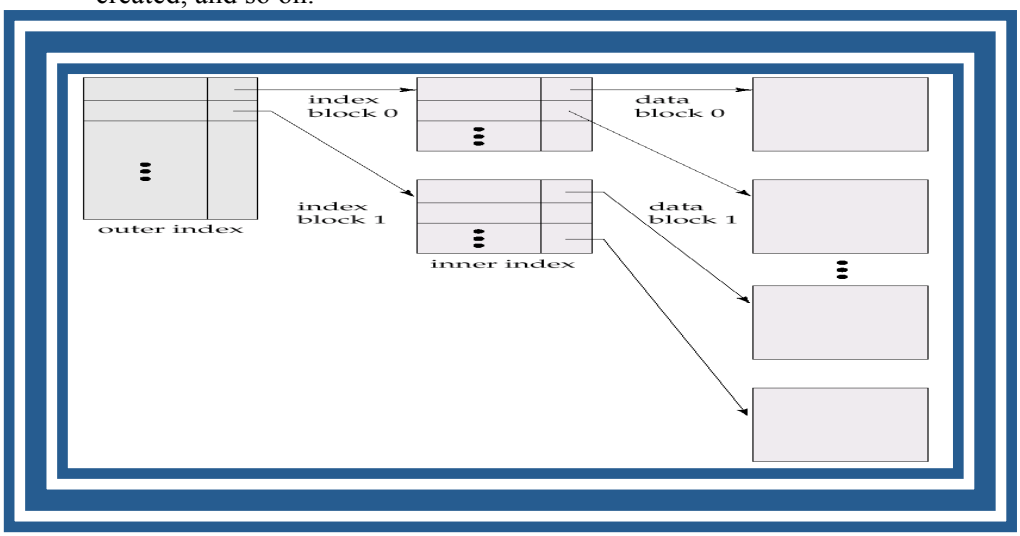
Secondary Indices

- Index record points to a bucket that contains pointers to all the actual records with that particular searchkey value.

- Secondary indices have to be dense

Multilevel index

- If primary index does not fit in memory, access becomes expensive.
- To reduce number of disk accesses to index records, treat primary index kept on disk as a sequential file and construct a sparse index on it.
- outer index – a sparse index of primary index
- inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.



B^+ TREES INDEXING CONCEPTS

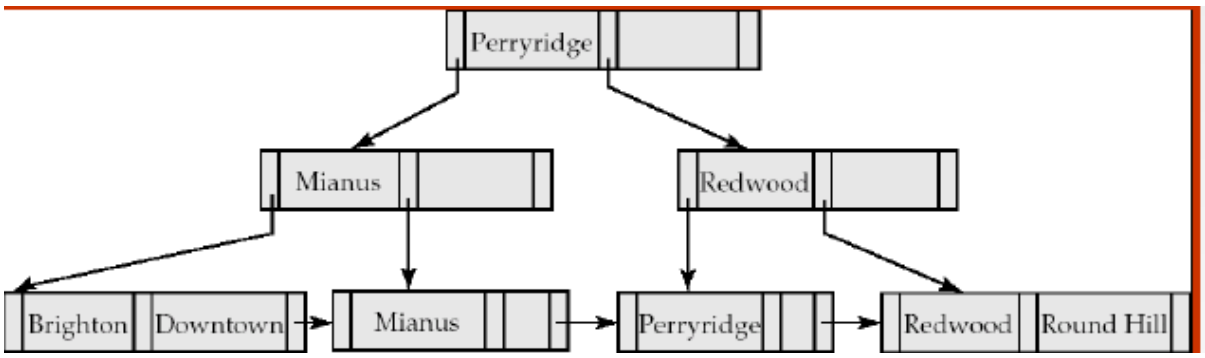
Disadvantage of indexed-sequential files: performance degrades as file grows, since many overflow blocks get created. Periodic reorganization of entire file is required.

Advantage of B^+-tree index files: automatically reorganizes itself with small, local, changes, in the face of insertions and deletions. Reorganization of entire file is not required to maintain performance.

Disadvantage of B^+-trees: extra insertion and deletion overhead, space overhead.

A B^+-tree is a rooted tree satisfying the following properties:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between [n/2] and n children.
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf, it can have between 0 and (n–1) values.



B+TREE NODE STRUCTURE

-Typical node

*Ki are the search-key values

*Pi are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).

-The search keys in a node are ordered

K1 < K2 < K3 < . . . < Kn–1

Properties of leaf node

- For i = 1, 2, . . ., n–1, pointer P_i either points to a file record with search-key value K_i, or to a bucket of pointers to file records, each record having search-key value K_i.
- P_n points to next leaf node in search-key order

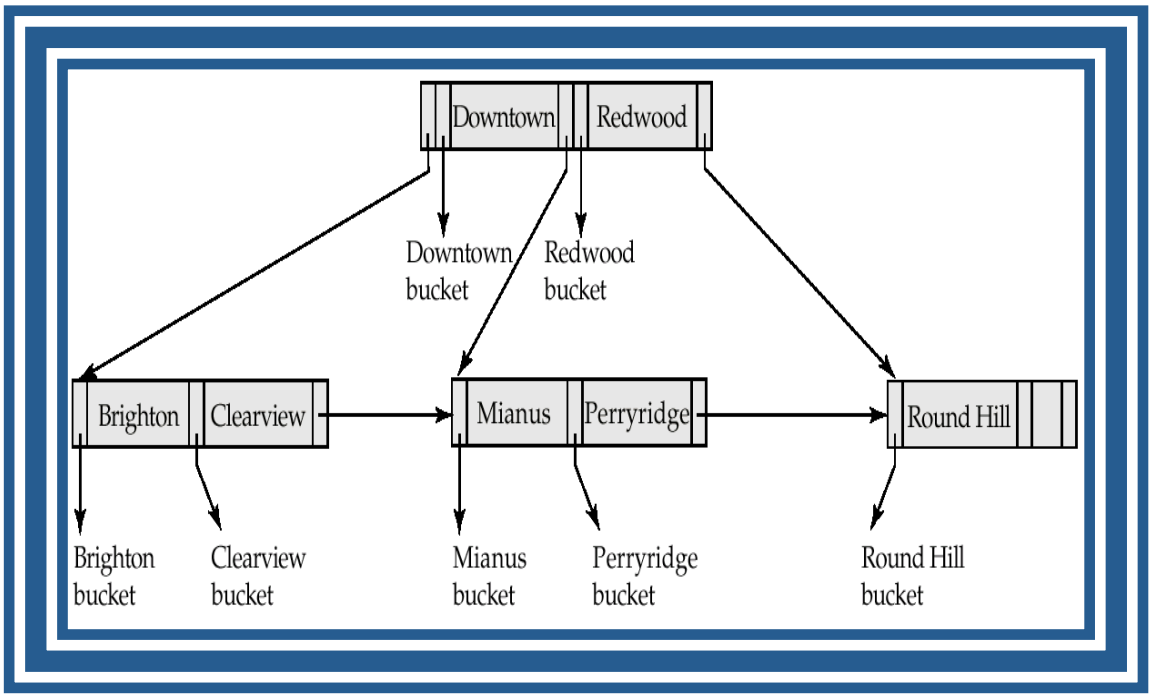
Non-Leaf Nodes in B^+-Trees

Non leaf nodes form a multi-level sparse index on the leaf nodes. For a non-leaf node with m pointers: All the search-keys in the sub tree to which P_1 points are less than K_1.

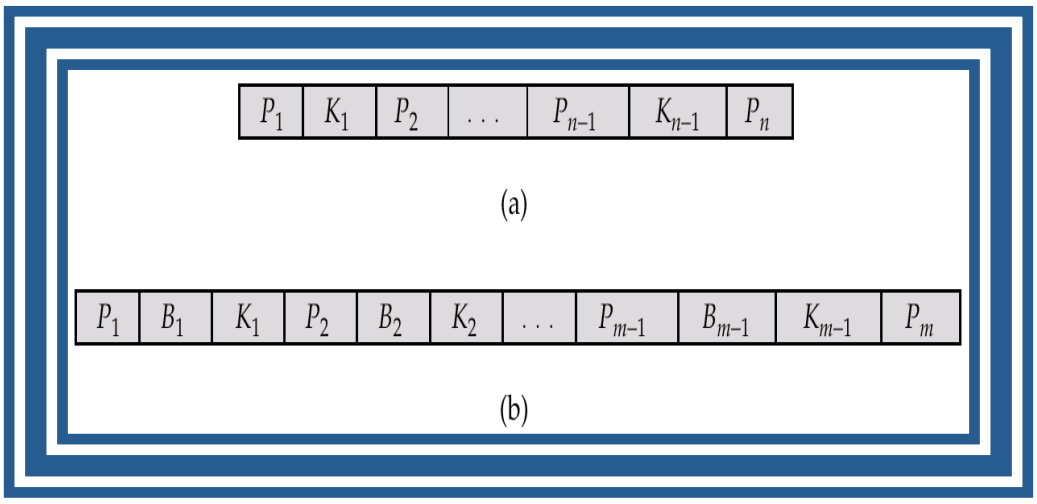
B TREES INDEXING CONCEPTS

-
- Similar to B+-tree, but B-tree allows search-key values to appear only once; eliminates redundant storage of search keys.
- Search keys in nonleaf nodes appear nowhere else in the B-tree; an additional pointer field for each search key in a nonleaf node must be included.

Generalized B-tree leaf node



Non leaf nodes – pointers Bi are the bucket or file record pointers.

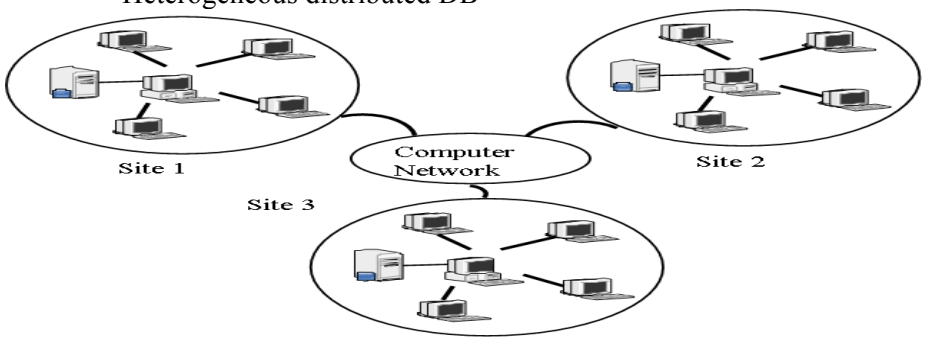


DISTRIBUTED DATABASES

In distributed database system data reside in several location where as centralized database system the data reside in single location

Classification

- Homogenous distributed DB
- Heterogeneous distributed DB



- Homogenous distributed DB
  - All sites have identical database management software, are aware of one another.
  - Agree to cooperate in processing users' request.
- Heterogeneous distributed DB
  - Different sites may use different schemas and different DBMS software.
  - The sites may not be aware of one another
  - Provide only limited facilities for cooperation in transaction processing.
- Consider a relation r; there are two approaches to store this relation in the distributed DB.
  - Replication
  - Fragmentation
- Replication
  - The system maintains several identical replicas (copies) of the relation at different site.
  - Full replication- copy is stored in every site in the system.
- Advantages and disadvantages
  - Availability
  - Increased parallelism

Increased overhead update

- Fragmentation
  - The system partitions the relation into several fragment and stores each fragment at different sites
  - Two approaches
    - Horizontal fragmentation
    - Vertical fragmentation

Horizontal fragmentation

Splits the relation by assigning each tuple of r to one or more fragments

relation r is partitioned into a number of subsets, $r_1, r_2, \ldots . r_n$ and can be reconstruct the original relation using union of all fragments, that is

$$r = r_1 \cup r_2 \cup \ldots\ldots \cup r_n$$

- Vertical fragmentation
  - Splits the relation by decomposing scheme R of relation and reconstruct the original relation by using natural join of all fragments. that is

$$r = r_1 \quad r_2 \quad \ldots\ldots \quad r_n$$

MULTIDIMENSIONAL AND PARALLEL DATABASE

MULTIDIMENSIONAL DATABASES:

A multidimensional database (MDB) is a type of database that is optimized for data warehouse and online analytical processing (OLAP) applications. Multidimensional databases are frequently created using input from existing relational databases. Whereas a relational database is typically accessed using a Structured Query Language (SQL) query, a multidimensional database allows a user to ask questions like "How many Aptivas have been sold in Nebraska so far this year?" and similar questions related to summarizing business operations and trends. An OLAP application that accesses data from a multidimensional database is known as a MOLAP (multidimensional OLAP) application.

A multidimensional database - or a multidimensional database management system (MDDBMS) - implies the ability to rapidly process the data in the database so that answers can be generated quickly. A number of vendors provide products that use multidimensional databases. Approaches to how data is stored and the user interface vary. Conceptually, a multidimensional database uses the idea of a data cube to represent the dimensions of data available to a user. For example, "sales" could be viewed in the dimensions of product model, geography, time, or some additional dimension. In this case, "sales" is known as the measure attribute of the data cube and the other dimensions are seen as feature attributes. Additionally, a database creator can define hierarchies and levels within a dimension (for example, state and city levels within a regional hierarchy).

PARALLEL DATABASES:

A parallel database is designed to take advantage of such architectures by running multiple instances which "share" a single physical database. In appropriate applications, a parallel server can allow access to a single database by users on multiple machines, with increased performance.

A parallel server processes transactions in parallel by servicing a stream of transactions using multiple CPUs on different nodes, where each CPU processes an entire transaction. Using parallel data manipulation language you can have one transaction being performed by multiple nodes. This is an efficient approach because many applications consist of online insert and update transactions which tend to have short data access requirements. In addition to balancing the workload among CPUs, the parallel database provides for concurrent access to data and protects data integrity.

Key elements of parallel processing:

- Speedup and Scale up: the Goals of Parallel Processing
- Synchronization: A Critical Success Factor
- Locking
- Messaging

MOBILE DATABASE

A **mobile database** is either a stationary database that can be connected to by a mobile computing device (e.g., smart phones and PDAs) over a mobile network, or a database which is actually stored by the mobile device. This could be a list of contacts, price information, distance travelled, or any other information

WEB DATABASES

Web Database is a web page API for storing data in databases that can be queried using a variant of SQL

MULTIMEDIA DBMS

A multimedia database management system (MM-DBMS) is a framework that manages different types of data potentially represented in a wide diversity of formats on a wide array of media sources.
Like the traditional DBMS, MM-DBMS should address requirements:

Integration
- Data items do not need to be duplicated for different programs

Data independence
- Separate the database and the management from the application programs

Concurrency control
- allows concurrent transactions

Requirements of Multimedia DBMS

Persistence
- Data objects can be saved and re-used by different transactions and program invocations

Privacy
- Access and authorization control

Integrity control
- Ensures database consistency between transactions

Recovery
- Failures of transactions should not affect the persistent data storage

Query support
- Allows easy querying of multimedia data

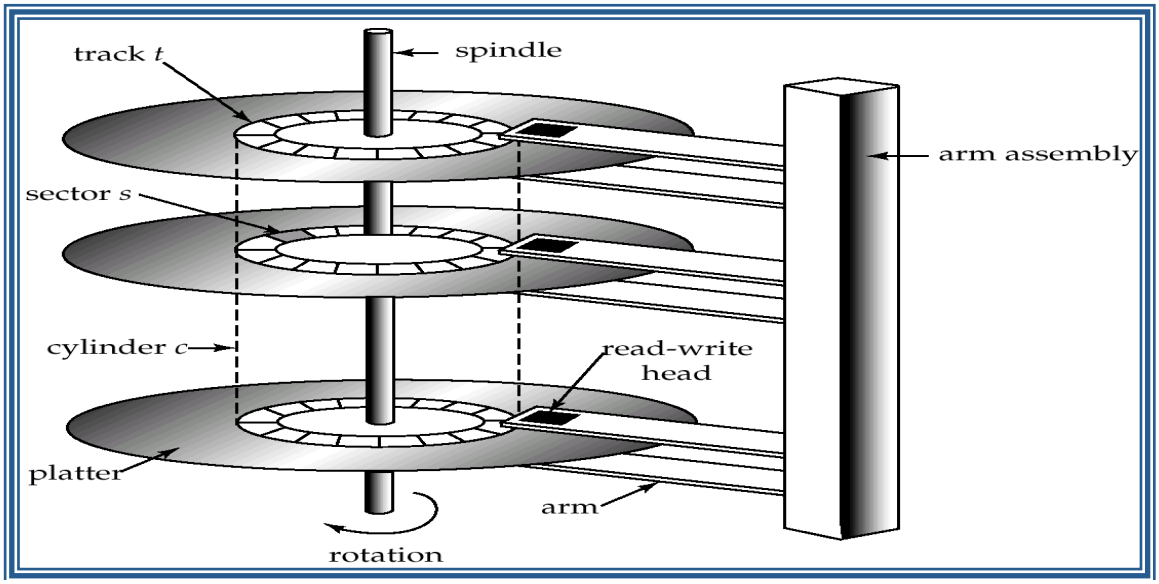
SPATIAL DATABASE

- A SDBMS is a DBMS
- It offers spatial data types/data models/ query language
  - Support spatial properties/operations
- It supports spatial data types in its implementation
  - Support spatial indexing, algorithms for spatial selection and join

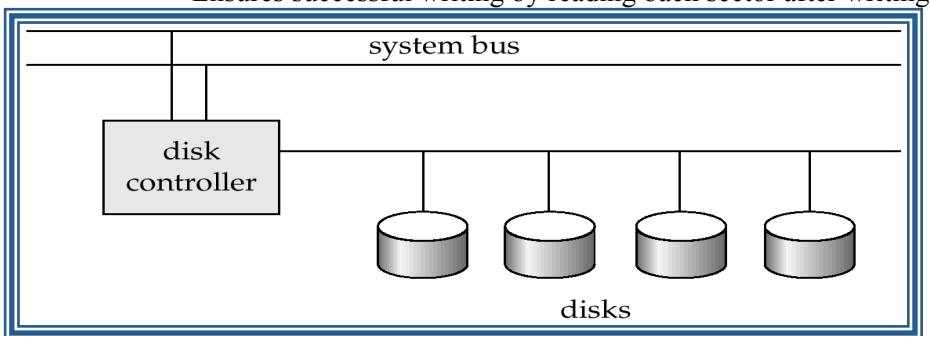
Spatial Database Applications

- GIS applications (maps):
  - Urban planning, route optimization, fire or pollution monitoring, utility networks, etc
- Other applications:
  - VLSI design, CAD/CAM, model of human brain, etc
- Traditional applications:
  - Multidimensional records

MAGNETIC HARD DISK MECHANISM



- **Read-write head**
  - Positioned very close to the platter surface
  - Reads or writes magnetically.
- Surface of platter divided into circular **tracks**
- Each track is divided into **sectors.**
  - A sector is the smallest unit of data that can be read or written.
- Head-disk assemblies
  - multiple disk platters on a single spindle (typically 2 to 4)
  - one head per platter, mounted on a common arm.
- **Cylinder** *i* consists of i^{th} track of all the platters
- **Disk controller** – interfaces between the computer system and the disk drive hardware.
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Ensures successful writing by reading back sector after writing it.



- Multiple disks connected to a computer system through a controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SCSI (Small Computer System Interconnect) range of standards