Nadar Saraswathi College of Engineering & Technology

NSCET E-LEARNING PRESENTATION

LISTEN … LEARN… LEAD…

COMPUTER SCIENCE AND ENGINEERING

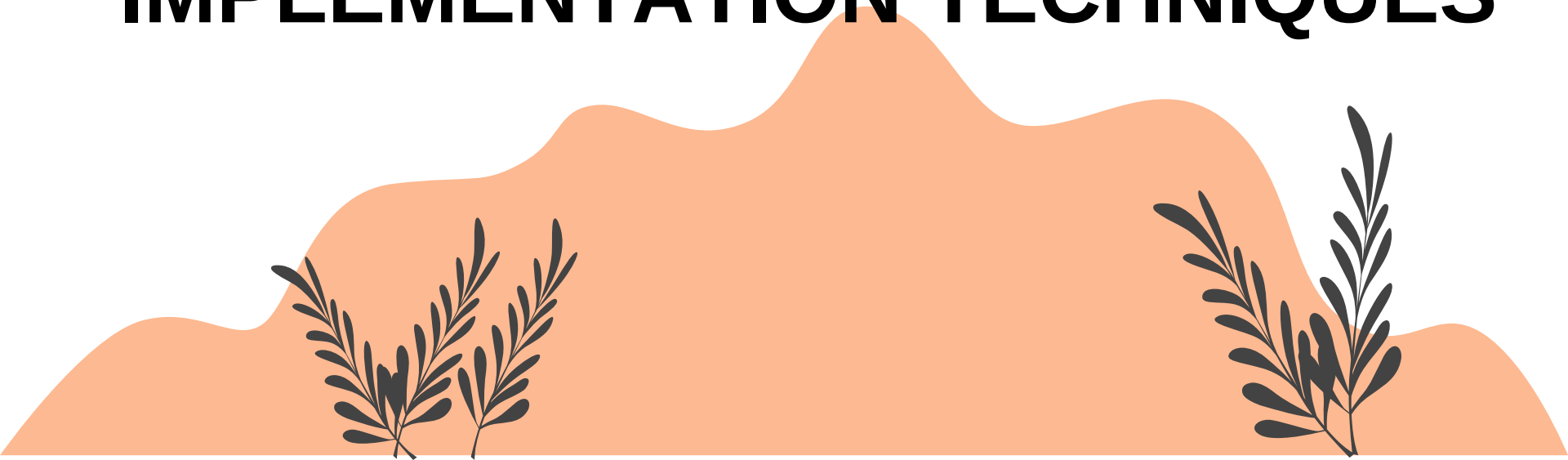
II YEAR / IV SEMESTER

CS8492 – DATABASE MANAGEMENT SYSTEMS

A.AMBIKA,M.E,MISTE
ASSISTANT PROFESSOR
Nadar Saraswathi College of Engineering & Technology,
Vadapudupatti, Annanji (po), Theni – 625531.

UNIT- IV

IMPLEMENTATION TECHNIQUES



CONTENTS

- RAID
- File Organization
- Organization of Records in Files
- Indexing and Hashing
- Ordered Indices
- B+ tree Index Files

CONTENTS

- B tree Index Files
- Static Hashing
- Dynamic Hashing
- Query Processing Overview
- Algorithms for SELECT and JOIN operations
- Query optimization using Heuristics and Cost Estimation

RAID

- RAID-redundant array of independent disks
- originally redundant array of inexpensive disks is a way of storing the same data in different places on multiple hard disks to protect data in the case of a drive failure.

RAID

Motivation for RAID:

- Just as additional memory in form of cache, can improve the system performance, in the same way additional disks can also improve system performance.
- In RAID we can use an array of disks which operates independently since there are many disks, multiple I/O requests can be handled in parallel if the data required is on separate disks.

RAID

- A single I/O operation can be handled in parallel ifthe data required is distributed across multiple disks.

Benefits of RAID:

- Data loss can be very dangerous for an organization
- RAID technology prevents data loss due to disk failure.
- RAID technology can be implemented in hardware or Software.
- Servers make use of RAID Technology

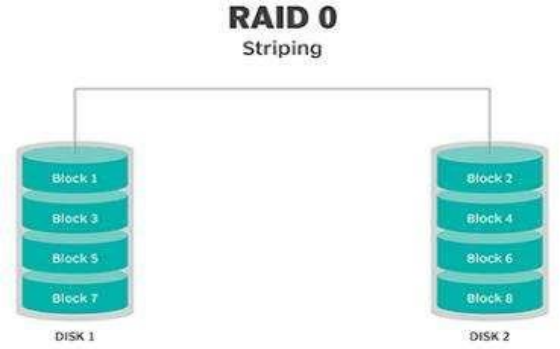
RAID

RAID level 0:

- It divides data into block units and writes them across a number of disks.
- As data is placed across multiple disks it is also called “data Striping”.
- The advantage of distributing data over disks is that if different I/O requests are pending for two different blocks of data, then there is a possibility that the requested blocks are on different disks

RAID

RAID LEVEL0:



Advantages:

- I/O performance is greatly improved by spreading the I/O load across many channels & drives.
- Best performance is achieved when data is striped across multiple controllers with only one driver per controller.

RAID

Disadvantages

- It is not fault-tolerant, failure of one drive will result in all data in an array being lost

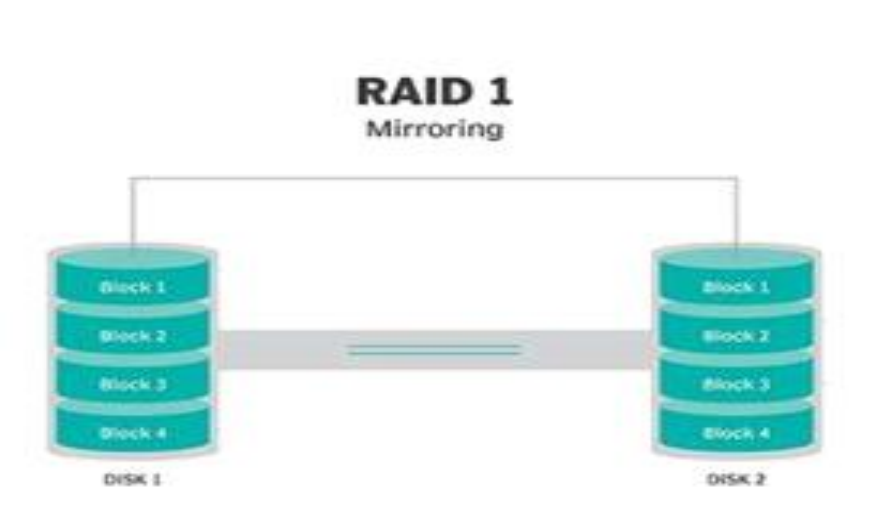
RAID

RAID Level 1: Mirroring (or shadowing):

- Also known as disk mirroring, this configuration consists of at least two drives that duplicate the storage of data,There is no striping.
- Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage.
- Every write is carried out on both disks. If one disk in a pair fails, data still available in the other.

RAID

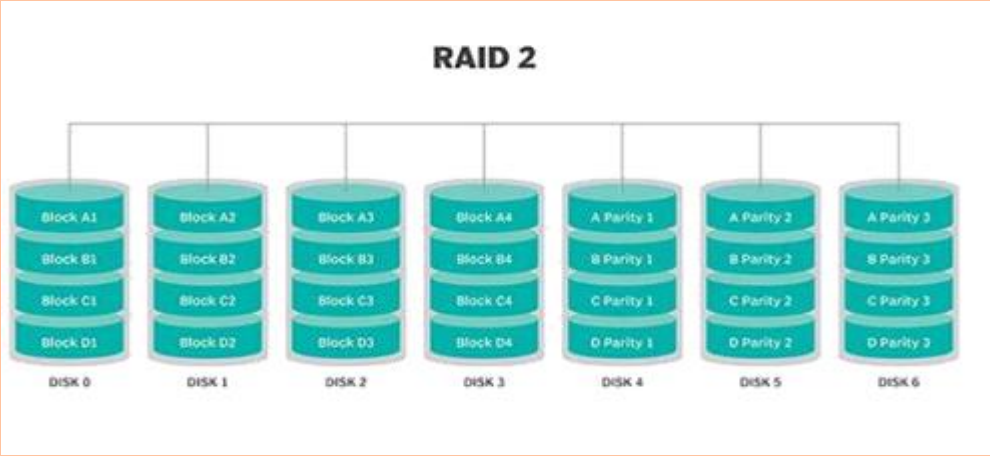
- Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired Probability of combined event is very small.



RAID

RAID Level 2:

- This configuration uses striping across disks, with some disks storing error checking and correcting (ECC) information. It has no advantage over RAID 3 and is no longer used.



RAID

RAID Level 3: Bit-Interleaved Parity

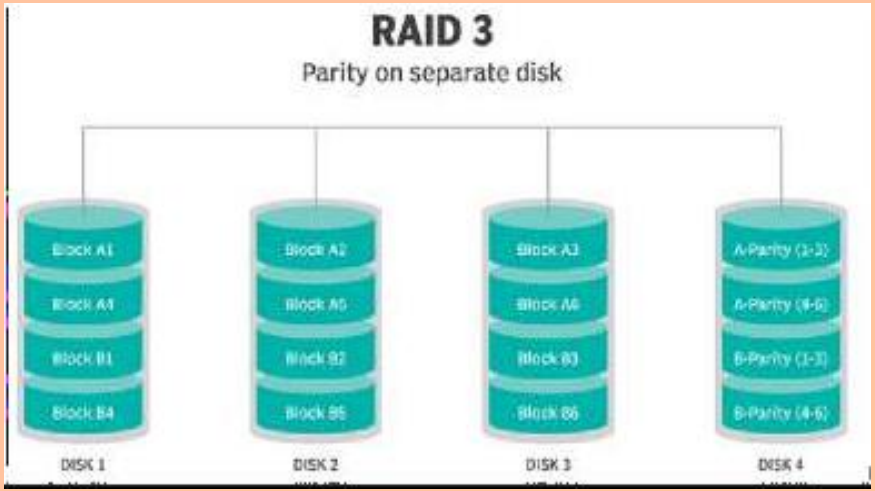
- A single parity bit is enough for error correction, not just detection, since we know which disk has failed

  – 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

- I/O operation addresses all the drives at the same time, RAID 3 cannot overlap I/O. For this reason, RAID 3 is best for single-user systems with long record applications.

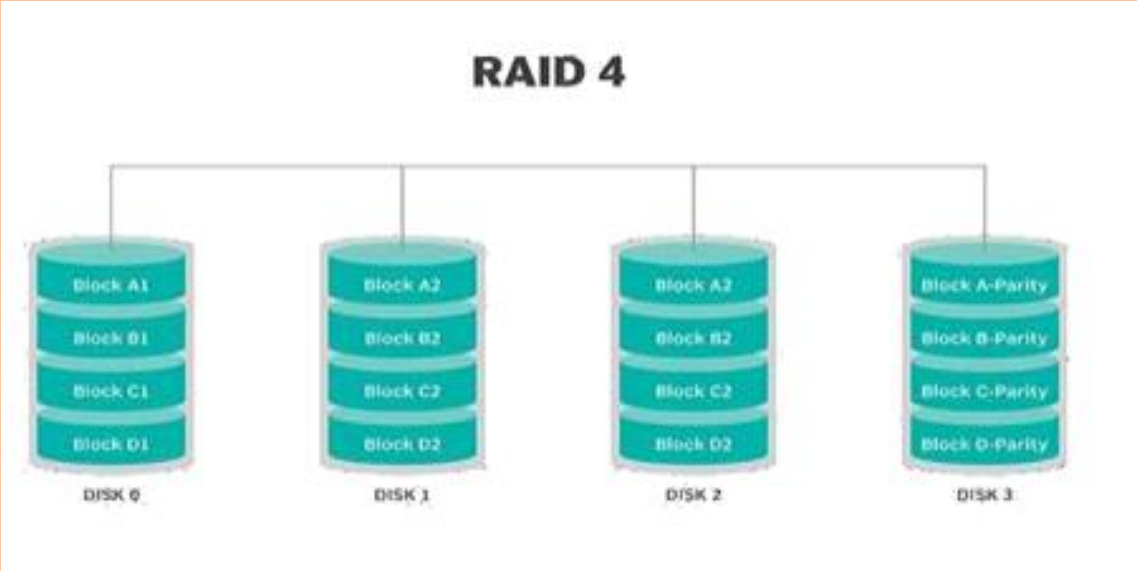


RAID

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

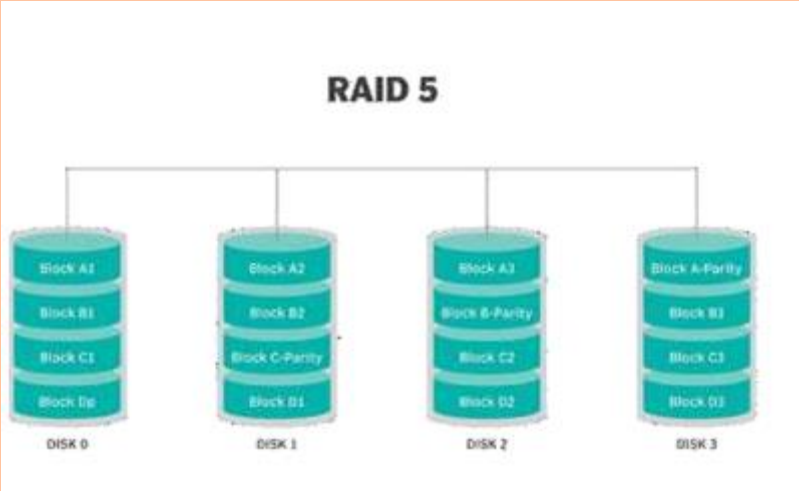

RAID 4
Block A1
Block B1
Block C1
Block D1
DISK 0
Block A2
Block B2
Block C2
Block D2
DISK 1
Block A2
Block B2
Block C2
Block D2
DISK 2
Block A-Parity
Block B-Parity
Block C-Parity
Block D-Parity
DISK 3

RAID

RAID Level 5:

- RAID 5 uses striping as well as parity for redundancy. It is well suited for heavy read and low write operations.
- 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

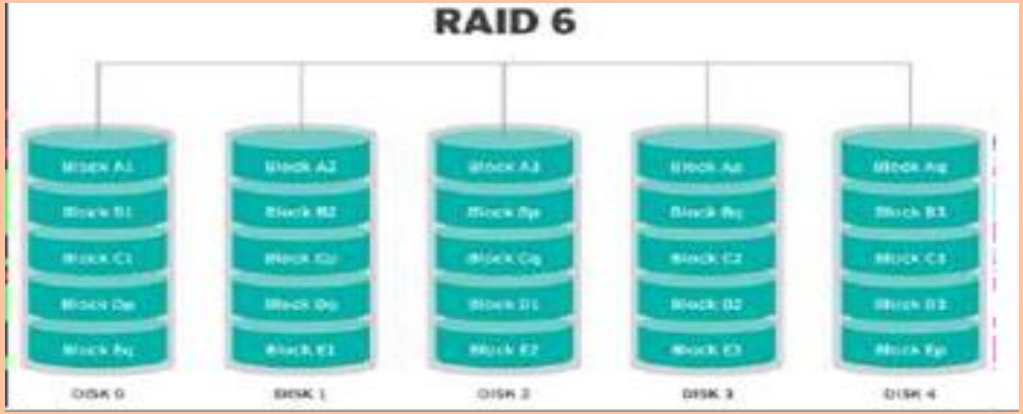

RAID 5
Block A1
Block B1
Block C1
Block Dp
Block A2
Block B2
Block C-Parity
Block D1
Block A3
Block B-Parity
Block C2
Block D2
Block A-Parity
Block B3
Block C3
Block D3
DISK 0
DISK 1
DISK 2
DISK 3

RAID Level 6:

- This technique is similar to RAID 5, but includes a second parity scheme that is distributed across the drives in the array.
- The use of additional parity allows the array to continue to function even if two disks fail simultaneously. However, this extra protection comes at a cost.

RAID

- 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.



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
  - Fixed length record
  - Variable length record

FILE ORGANIZATION

Fixed length record approach:

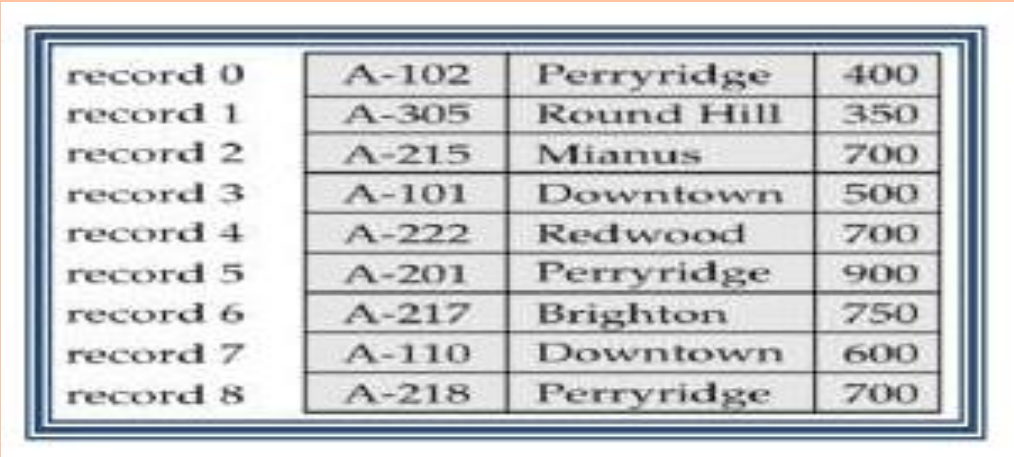
Assume record size is fixed each file has records of one particular type only different files are used for different relations.

Simple approach:

Example pseudo code

```
type account = record
    account_number char(10);
    branch_name char(22);
    balance numeric(8);
    end
```

FILE ORGANIZATION

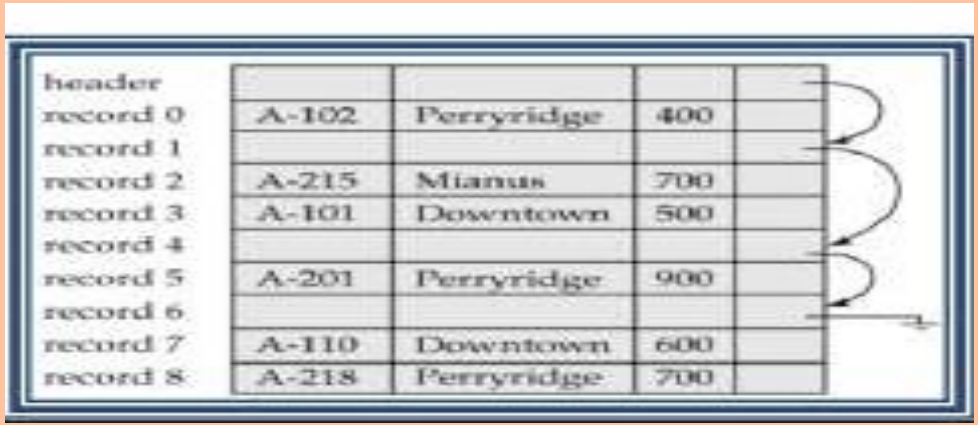


record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

FILE ORGANIZATION

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



FILE ORGANIZATION

Variable-Length Records:

- Attach an end-of-record.
- control character to the end of each record.
- Difficulty with deletion

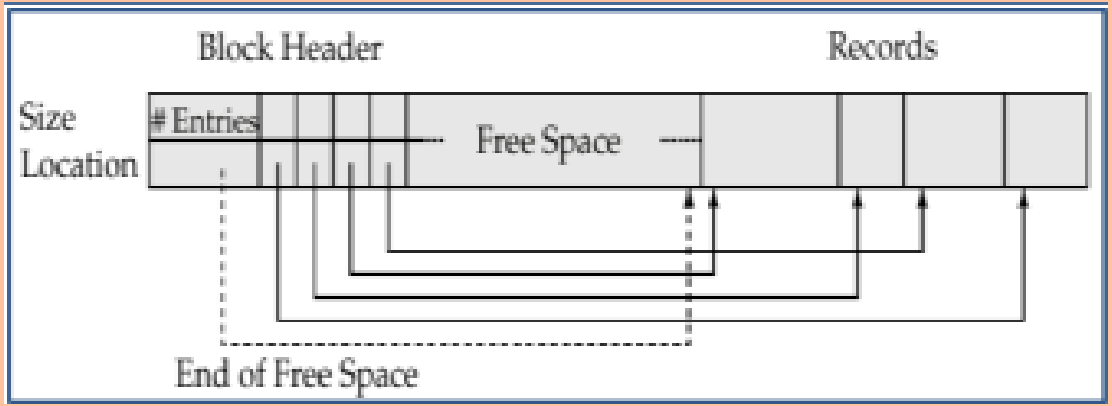
Disadvantage:

- It is not easy to reuse space occupied formerly by deleted record.
- There is no space in general for records grows longer

FILE ORGANIZATION

Slotted Page Structure

- Slotted page header contains:
  - Number of record entries
  - End of free space in the block
  - Location and size of each record



FILE ORGANIZATION

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.

Disadvantage:

Space is wasted in all records except the first in a a chain.

FILE ORGANIZATION

Solution is to allow two kinds of block in file:

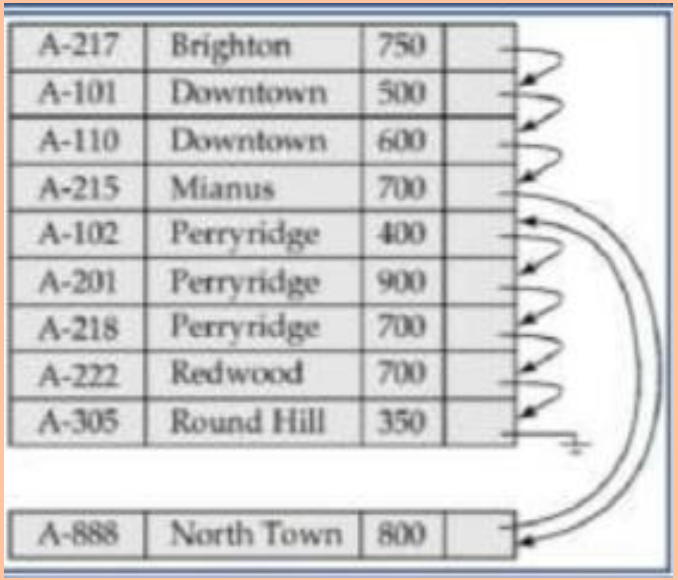
- Anchor block – contains the first records of chain.
- Overflow block – contains records other than those that are the first records of chains

ORGANIZATION OF RECORDS IN FILES

- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Heap** – a record can be placed anywhere in the file where there is space
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

ORGANIZATION OF RECORDS IN FILES

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	



A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	
A-888	North Town	800	

- Deletion –Use pointer chains.

ORGANIZATION OF RECORDS IN FILES

Insertion – locate the position where the record is to be inserted – if there is free space insert there

– if no free space, insert the record in an overflow block.

– In either case, pointer chain must be updated.

INDEXING AND HASHING

Basic Concepts:

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- Search Key - attribute to set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form Search-key pointer.
- Index files are typically much smaller than the original file.

INDEXING AND HASHING

- Two basic kinds of indices:
  – Ordered indices: search keys are stored in sorted order
  – Hash indices: search keys are distributed uniformly across “buckets “and by using a “hash function” the values are determined.
- In an ordered index, index entries are stored sorted on the search key value.

INDEXING AND HASHING

- 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.
    - Dense index -Index record appears for every search-key value in the file.
    - Sparse index-Contains index records for only some search-key values.

INDEXING AND HASHING

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.

INDEXING AND HASHING

Index Update:

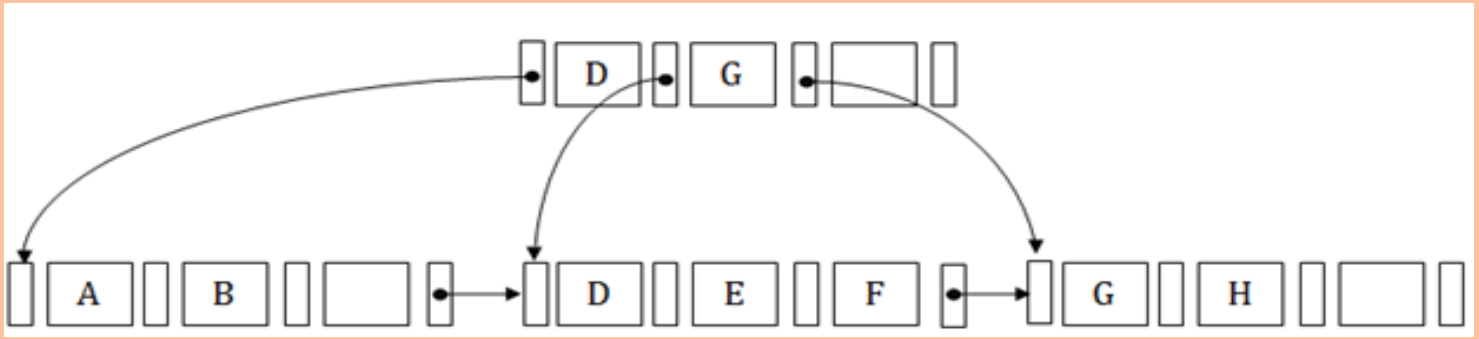
Deletion If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.

-Single-level index deletion

-Single-level index insertion

B + TREE INDEX FILES

Srtucture of B+ Tree:



- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.

B + TREE INDEX FILES

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

B + TREE INDEX FILES

Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

Leaf node

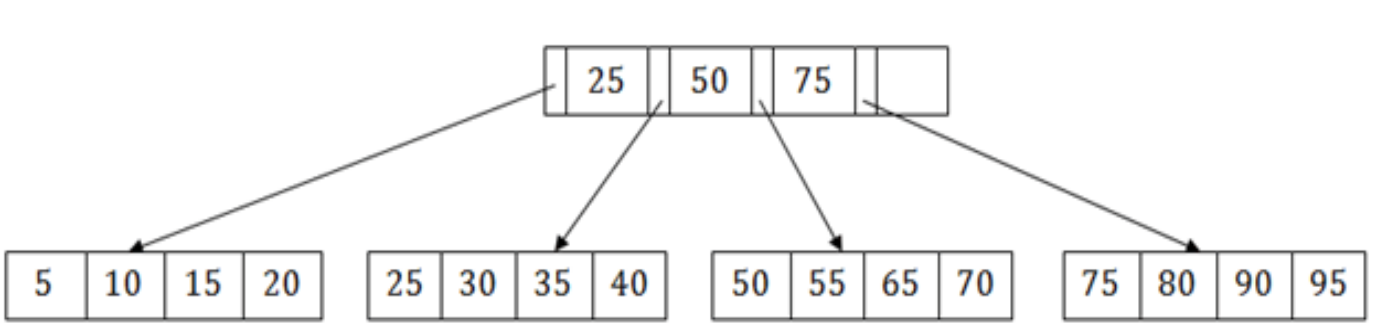
- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

B + TREE INDEX FILES

Searching a record in B+ Tree:

- Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.
- So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.

B + TREE INDEX FILES

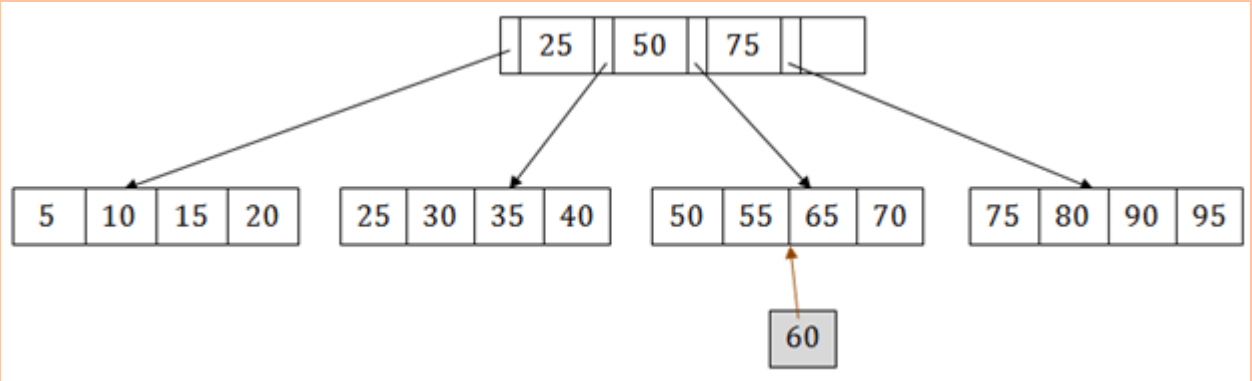


B+ Tree Insertion:

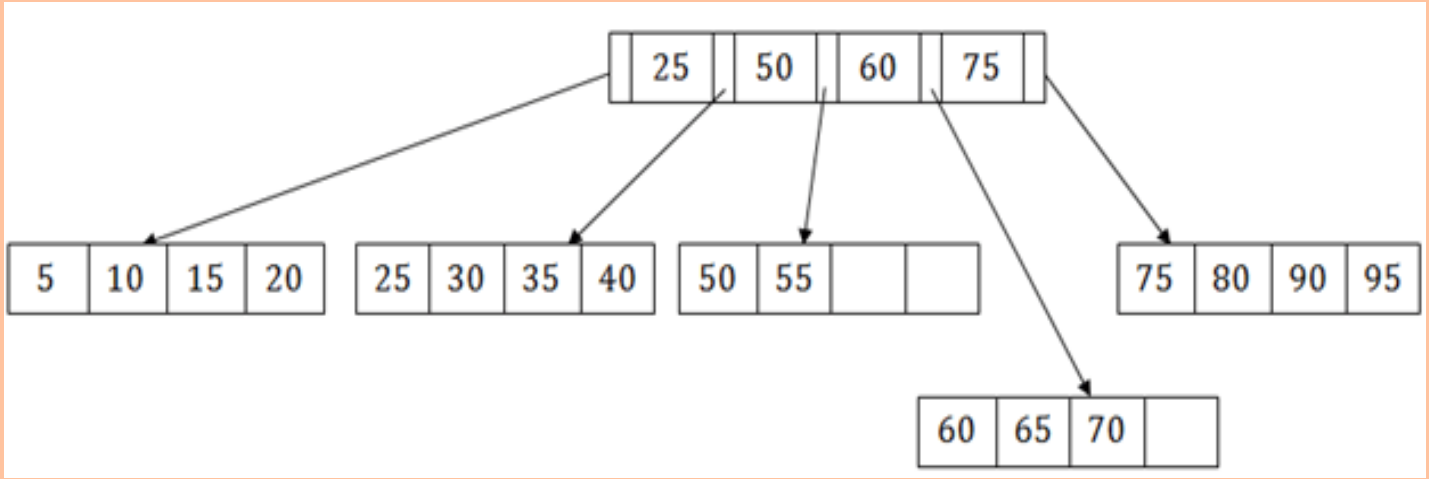
- Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

B + TREE INDEX FILES

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



B + TREE INDEX FILES



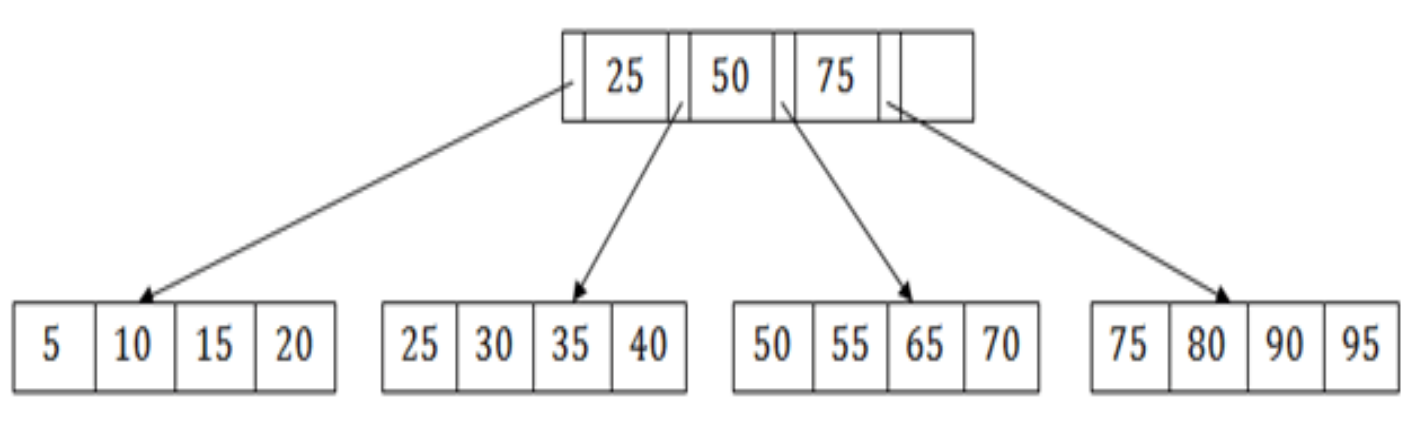
B + TREE INDEX FILES

B+ Tree Deletion:

- Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too.
- If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

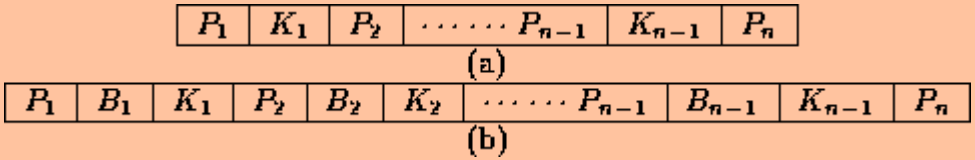
B + TREE INDEX FILES

- After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:



B TREE INDEX FILES

- B-tree indices are similar to B+-tree indices.
- Difference is that B-tree eliminates the redundant storage of search key values.
- In B-tree of some search key values appear twice.
- A corresponding B-tree of allows search key values to appear only once.
- Thus we can store the index in less space.



B TREE INDEX FILES

Advantages:

•Lack of redundant storage (but only marginally different).

•Some searches are faster (key may be in non-leaf node).

Disadvantages:

•Leaf and non-leaf nodes are of different size(complicates storage).

•Deletion may occur in a non-leaf node (more complicated).

•Generally, the structural simplicity of B-tree is preferred.

HASHING

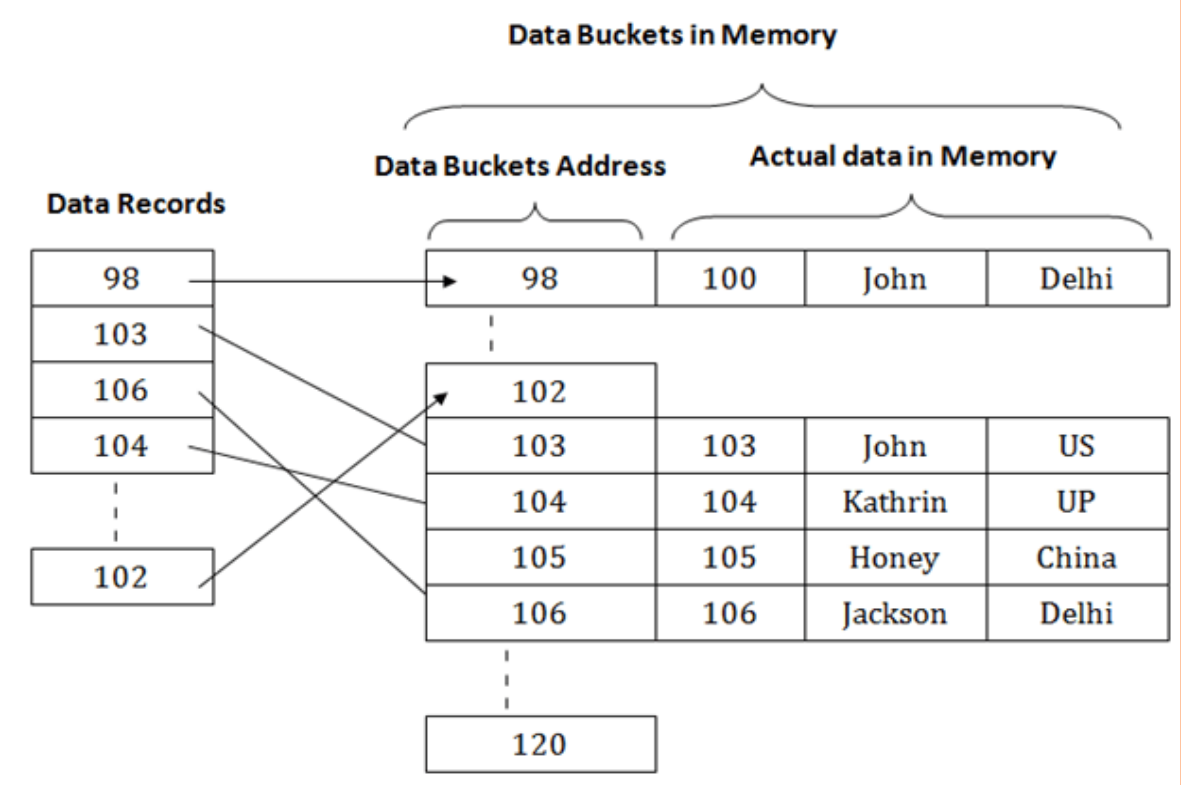
Hashing is a technique to directly search the location of desired data on the disk without using index structure.

Hashing method is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value.

Types of Hashing:

1. Static hashing
2. Dynamic hashing

HASHING


Data Buckets in Memory
Data Buckets Address
Actual data in Memory
Data Records
98
103
106
104
102
98
100
John
Delhi
102
103
103
John
US
104
104
Kathrin
UP
105
105
Honey
China
106
106
Jackson
Delhi
120

STATIC HASHING

- In **static hashing**, when a search-key value is p rovided, the hash function always computes the same address.
- For example, if mod-4 hash function is used, then it shall generate only 5 values.
- The output address shall always be same for that function.
- The number of buckets provided remains unchanged at all times.

STATIC HASHING

Operations of Static Hashing:

- Searching a record
- Insert a Record
- Delete a Record
- Update a Record

Bucket Overflow.

If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address.

STATIC HASHING

This is a critical situation in this method. To overcome this situation, there are various methods. Some commonly used methods are as follows:

1.Open Hashing
2.Close Hashing

DYNAMIC HASHING

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

DYNAMIC HASHING

How to search a key

- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

DYNAMIC HASHING

How to insert a new record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

DYNAMIC HASHING

Advantages of dynamic hashing

- The performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- Memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- Good for the dynamic database where data grows and shrinks frequently.

DYNAMIC HASHING

Disadvantages of dynamic hashing

- If the data size increases then the bucket size is also increased.
- The bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

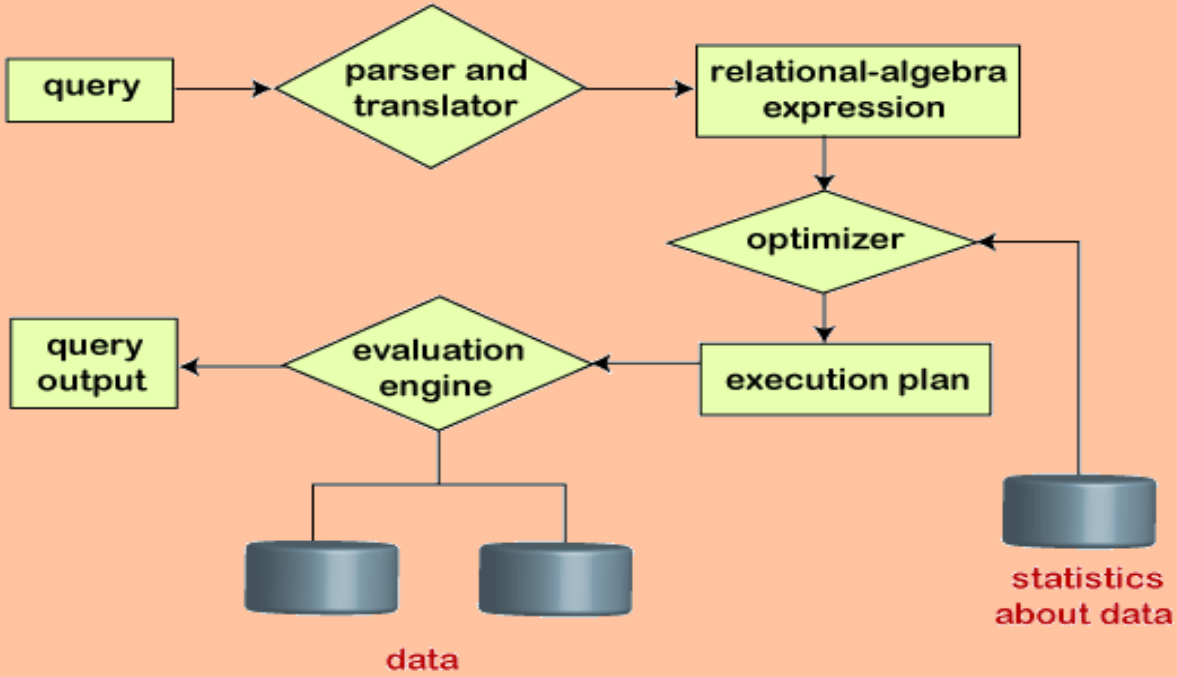
QUERY PROCESSING IN DBMS

- Query Processing is the activity performed in extracting data from the database.
- In query processing, it takes various steps for fetching the data from the database.

The steps involved are:

1. Parsing and translation
2. Optimization
3. Evaluation

QUERY PROCESSING IN DBMS



Steps in query processing

MEASURE OF QUERY COST

- Cost is generally measured as total elapsed time for answering query
- Many factors contribute to time cost "disk accesses, CPU, or even network communication
- Typically disk access is the predominant cost, and is also relatively easy to estimate.
  - Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost

MEASURE OF QUERY COST

- Cost to write a block is greater than cost to read a block .
  – data is read back after being written to ensure that the write was successful.
- For simplicity we just use number of block transfers from disk as the cost measure.
- We ignore the difference in cost between sequential and random I/O for simplicity.
- We also ignore CPU costs for simplicity

MEASURE OF QUERY COST

- Costs depends on the size of the buffer in main memory.
- Having more memory reduces need for disk access.
- Amount of real memory available to buffer depends on other concurrent OS processes, and hard to determine ahead of actual execution.
- We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available.

MEASURE OF QUERY COST

- Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account.
- We do not include cost to writing output to disk in our cost formulae

ALGORITHMS FOR SELECT AND JOIN OPERATIONS

Basically file scan is a based on searching algorithms.These searching algorithms locate and retrieve the records that fulfils a selection condition.

1.Linear Search

2.Binary Search

ALGORITHMS FOR SELECT AND JOIN OPERATIONS

Linear Search:

- Search each file block and test all record to see whether they satisfy the selection condition.

Cost=b_r block transfers+1 seek

b_r-denotes number of blocks containing records from relation r.

- If selection is on a key attributes,can stop on finding record

Cost=(b_r /2)block transfers+1

ALGORITHMS FOR SELECT AND JOIN OPERATIONS

Advantage of Linear search:

- Linear search works even if there is no selection condition specified.
- For linear search,there is no need to have records in the file ordered form.
- Linear search works regardless of indices.

ALGORITHMS FOR SELECT AND JOIN OPERATIONS

Binary Search:

- Applicable if selection is an equality comparison on the attribute on which file is ordered.
- Assume that the blocks of a relation are sorted contiguously.
- Cost satimate is nothing but the number of disk blocks to be scanned.cost of locating the first tuple by a binary search on the blocks=$[\log_2(b_r)]*(t_T+t_s)$

ALGORITHMS FOR SELECT AND JOIN OPERATIONS

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SALARY > (SELECT MAX(SALARY) FROM EMPLOYEE WHERE DNO=5);

QUERY OPTIMIZATION USING HEURISTICS AND COST ESTIMATION

Heuristic-Based Query Optimization:

1. Break up SELECT operations with conjunctive conditions into a cascade of SELECT operations.
2. Using the commutativity of SELECT with other operations, move each SELECT operation as far down the query tree as is permitted by the attributes involved in the select condition.
3. Using commutativity and associativity of binary operations, rearrange the leaf nodes of the tree

QUERY OPTIMIZATION USING HEURISTICS AND COST ESTIMATION

4. Combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation in the tree into a JOIN operation, if the condition represents a join Condition.
5. Using the cascading of PROJECT and the commuting of PROJECT with other operations, break down and move lists of projection attributes down the tree as far as possible by creating new PROJECT operations as needed.

QUERY OPTIMIZATION USING HEURISTICS AND COST ESTIMATION

6.Identify sub-trees that represent groups of operations that can be executed by a single algorithm.

Cost Estimation in Query Optimization:

- The query optimizer should not depend solely on heuristic rules, but, it should also estimate the cost of executing the different strategies and find out the strategy with the minimum cost estimate.
- The cost functions used in query optimization are estimates and not exact cost functions.

QUERY OPTIMIZATION USING HEURISTICS AND COST ESTIMATION

- The cost of an operation is heavily dependent on its selectivity, that is, the proportion of select operation(s) that forms the output.
- In general the different algorithms are suitable for low or high selectivity queries.
- In order for query optimizer to choose suitable algorithm for an operation an estimate of the cost of executing that algorithm must be provided.

THANK YOU