DATABASE MANAGEMENT SYSTEMS

UNIT IV – IMPLEMENTATION TECHNIQUES

(Question Bank – Answers)

Part – A

1. **List all levels of RAID**

Ans.

- RAID-0 (Stripping)
- RAID-1 (Mirroring)
- RAID-2 (Bit-Level Stripping with Dedicated Parity)
- RAID-3 (Byte-Level Stripping with Dedicated Parity)
- RAID-4 (Block-Level Stripping with Dedicated Parity)
- RAID-5 (Block-Level Stripping with Distributed Parity)
- RAID-6 (Block-Level Stripping with two Parity Bits)

2. **Define ordered indices with example.**

Ans. The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example:

Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

3. **Distinguish between dense index and sparse index.**

Ans.

DENSE INDEX	**SPARSE INDEX**
• An index record appears for every search key value in file. • Indicies are faster. • Indicies requires more space. • Indicies impose more maintainance for insertions and deletions.	• An index record are created only for some of the records. • Indicies are slower • Indicies requires less space • Indicies improse less maintainance for insertions and deletions.

4. **Distinguish between primary index and secondary index**

Ans.

PRIMARY INDEX	**SECONDARY INDEX**
• Only one for the table. • Physical mechanism effect data distribution. • Can't be drop • Provides access path to the row • Storage and retrieval • Single amp operation	• Max 32 for a table • Logical mechanism doesn't provide access path to the row • Can be drop • Does not provide access path to the row • Only retrival • 2 or many amp operation

5. **Why is a B+ tree usually preferred as an access structure to a data file?**

Ans. In B+trees, data stored on the leaf node makes the search more efficient since we can store more keys in internal nodes – this means we need to access fewer nodes. Deleting data from a B+tree is easier and less time consuming because we only need to remove data from leaf nodes.

6. What are disadvantages of B Tree over B+ Tree?

Ans. Insertion in B tree is more complicated than B+ tree. B+ trees store redundant search keys but B tree has no redundant value. In a B+ tree, leaf node data is ordered as a sequential linked list but in a B tree the leaf node cannot be stored using a linked list.

7. What is a hash function? Give an example.

Ans. A Hash Function is a function that converts a given numeric or alphanumeric key to a small practical integer value. The mapped integer value is used as an index in the hash table.

Example:

A hash of the string "Hello world!" is "Hel"

8. What is the difference between dynamic and static hashing?

Ans.

DYNAMIC HASHING	**STATIC HASHING**
• The number of buckets are fixed. • Chaining is used. • Space overhead is more. • The bucket address table is not required. • The bucket is directly accessed.	• The number of buckets are not fixed. • There is no need of chaining. • Minimum space overhead. • The bucket address table is required. • The bucket address table is used to access the bucket.

9. List the costs involved in query execution.

Ans.

- Parsing
- Optimization
- Memory
- Input/Output
- CPU
- Network costs

10 . Outline the steps involved in query processing.

Ans.

- **Parsing and Translation :-** query is checked for syntax errors
- **Optimization :-** most efficient way to execute the given query.
- **Evaluation :-** operations on the data

Part – B

1. **Define RAID and explore the diversity of RAID levels, delving into their respective functionalities and characteristics in detail.**

Ans.

- **RAID** refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used.
- **RAID LEVELS:**

  There are 7 levels of RAID schemes. These schemas are as RAID 0, RAID 1, , RAID 2, , RAID 3, , RAID 4, , RAID 5, RAID 6.

  - **RAID 0**

    - RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
    - This level doesn't provide fault tolerance but increases the system performance

    Example:

Disk 0	**Disk 1**	**Disk2**	**Disk 3**
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35

In this figure, block 0, 1, 2, 3 form a stripe

In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one

Disk 0	**Disk 1**	**Disk2**	**Disk 3**
20	22	24	26
21	23	25	27
28	30	32	34
29	31	33	35

In this above figure, there is no duplication of data. Hence, a block once lost cannot be recovered

➢ RAID 1

- This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.

Example:

Disk 0	**Disk 1**	**Disk2**	**Disk 3**
A	A	B	B
C	C	D	D
E	E	F	F
G	G	H	H

Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

RAID 2

- RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.

- Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

RAID 3

- RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
- In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

Disk 0	**Disk 1**	**Disk2**	**Disk 3**
A	B	C	P(A, B, C)
D	E	F	P(D, E, F)
G	H	I	P(G, H, I)
J	K	L	P(J, K, L)

➢ RAID 4

- RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
- This level allows recovery of at most 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
- Level 3 and level 4 both are required at least three disks to implement RAID.

Disk 0	**Disk 1**	**Disk2**	**Disk 3**
A	B	C	P0
D	E	F	P1
G	H	I	P2
J	K	L	P3

- In this figure, we can observe one disk dedicated to parity.

- In this level, parity can be calculated using an XOR function. If the data bits are 0,0,0,1 then the parity bits is XOR(0,1,0,0) = 1. If the parity bits are 0,0,1,1 then the parity bit is XOR(0,0,1,1)= 0. That means, even number of one results in parity 0 and an odd number of one results in parity 1.

➢ RAID 5

- RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.
- It consists of block-level striping with DISTRIBUTED parity.
- Same as RAID 4, this level allows recovery of at most 1 disk failure. If more than one disk fails, then there is no way for data recovery.

Disk 0	Disk 1	Disk2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

- This figure shows that how parity bit rotates.
- This level was introduced to make the random write performance better.

➢ RAID 6

- This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
- In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5, and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another

disk fails then you won't be able to recover any of the data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

Disk 1	**Disk 2**	**Disk 3**	**Disk 4**
A0	B0	Q0	P0
A1	Q1	P1	P1
Q2	P2	C2	D2
P3	B3	C3	Q3

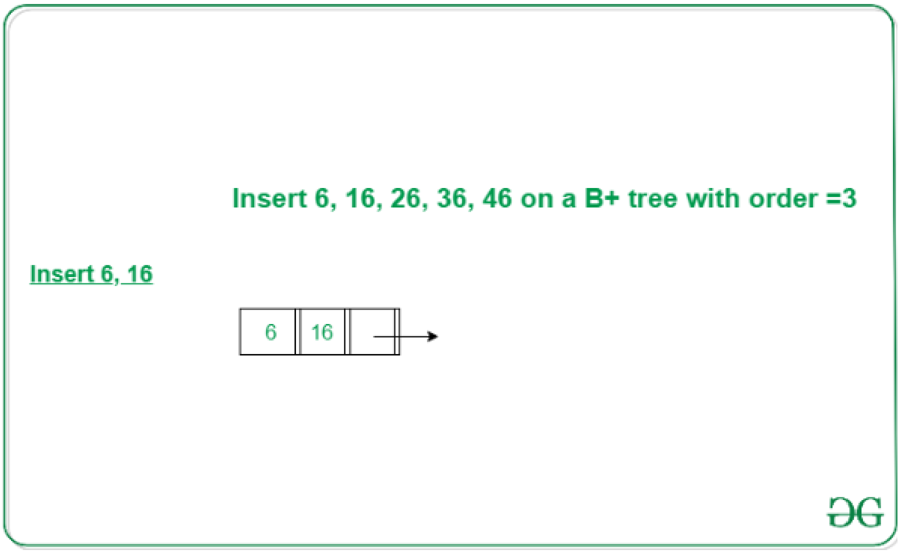
❖ CHARACTERISTICS:

- It contains a set of physical disk drives
- n this technology, the operating system views these separate disks as a single logical disk.
- In this technology, data is distributed across the physical drives of the array.
- Redundancy disk capacity is used to store parity information
- In case of disk failure, the parity information can be helped to recover the data.

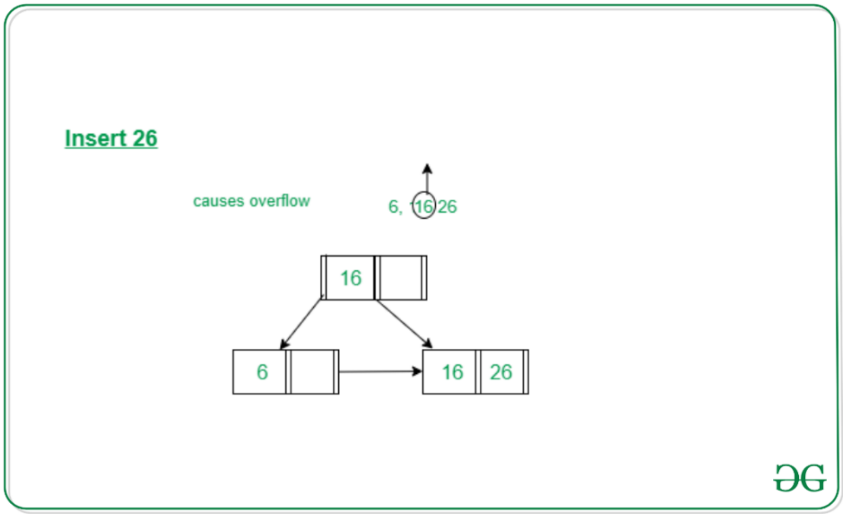
2. Sketch the structure of a B+ tree and give the algorithm for search in the B+ tree. Also insert the following key values 6, 16, 26, 36, 46 on a B+ tree with order = 3.

Ans.

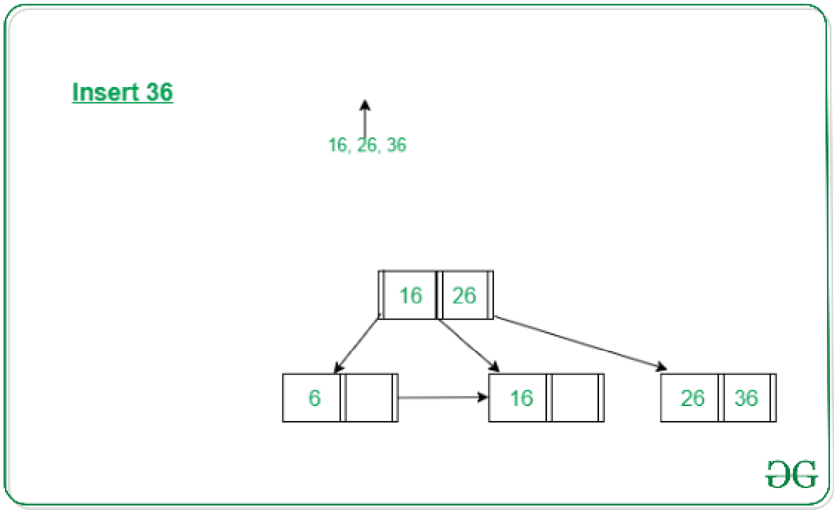
Step 1: The order is 3 so at maximum in a node so there can be only 2 search key values. As insertion happens on a leaf node only in a B+ tree so insert search key value **6 and 16** in increasing order in the node. Below is the illustration of the same:



Step 2: We cannot insert **26** in the same node as it causes an overflow in the leaf node, We have to split the leaf node according to the rules. First part contains **ceil((3-1)/2)** values i.e., only **6**. The second node contains the remaining values i.e., **16** and **26**. Then also copy the smallest search key value from the second node to the parent node i.e., **16** to the parent node. Below is the illustration of the same:



Step 3: Now the next value is **36** that is to be inserted after **26** but in that node, it causes an overflow again in that leaf node. Again follow the above steps to split the node. First part contains **ceil((3-1)/2)** values i.e., only **16**. The second node contains the remaining values i.e., **26** and **36**. Then also copy the smallest search key value from the second node to the parent node i.e., **26** to the parent node. Below is the illustration of the same:
The illustration is shown in the diagram below.



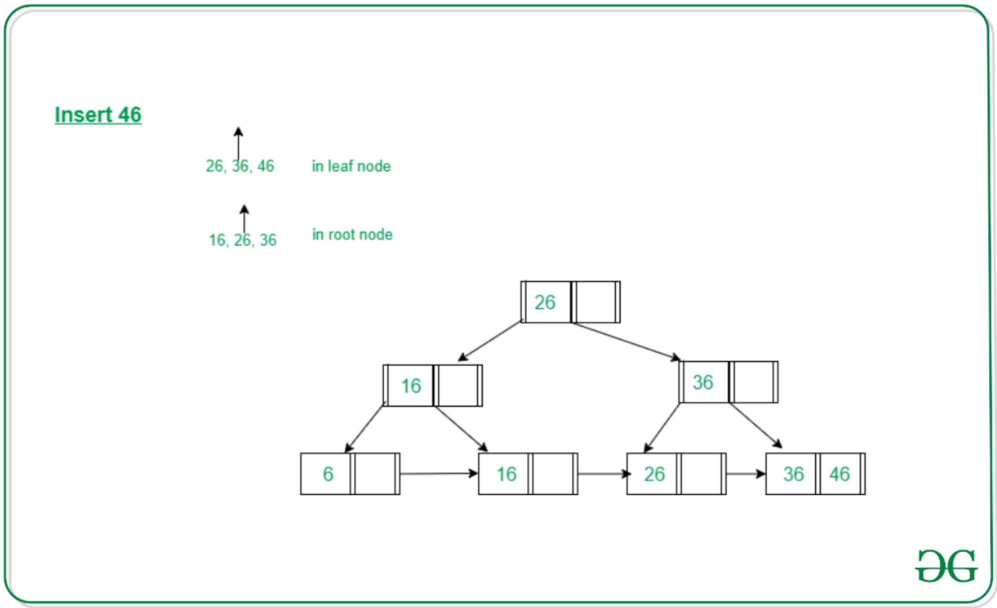
Step 4: Now we have to insert 46 which is to be inserted after **36** but it causes an overflow in the leaf node. So we split the node according to the rules. The first part contains **26** and the second part contains **36** and **46** but now we also have to copy **36** to the parent node but it causes overflow as only two search key values can be accommodated in a node. Now follow the steps to deal with overflow in the non-leaf node.

First node contains ceil(3/2)-1 values i.e. ’16’.

Move the smallest among remaining to the parent i.e ’26’ will be the new parent node.

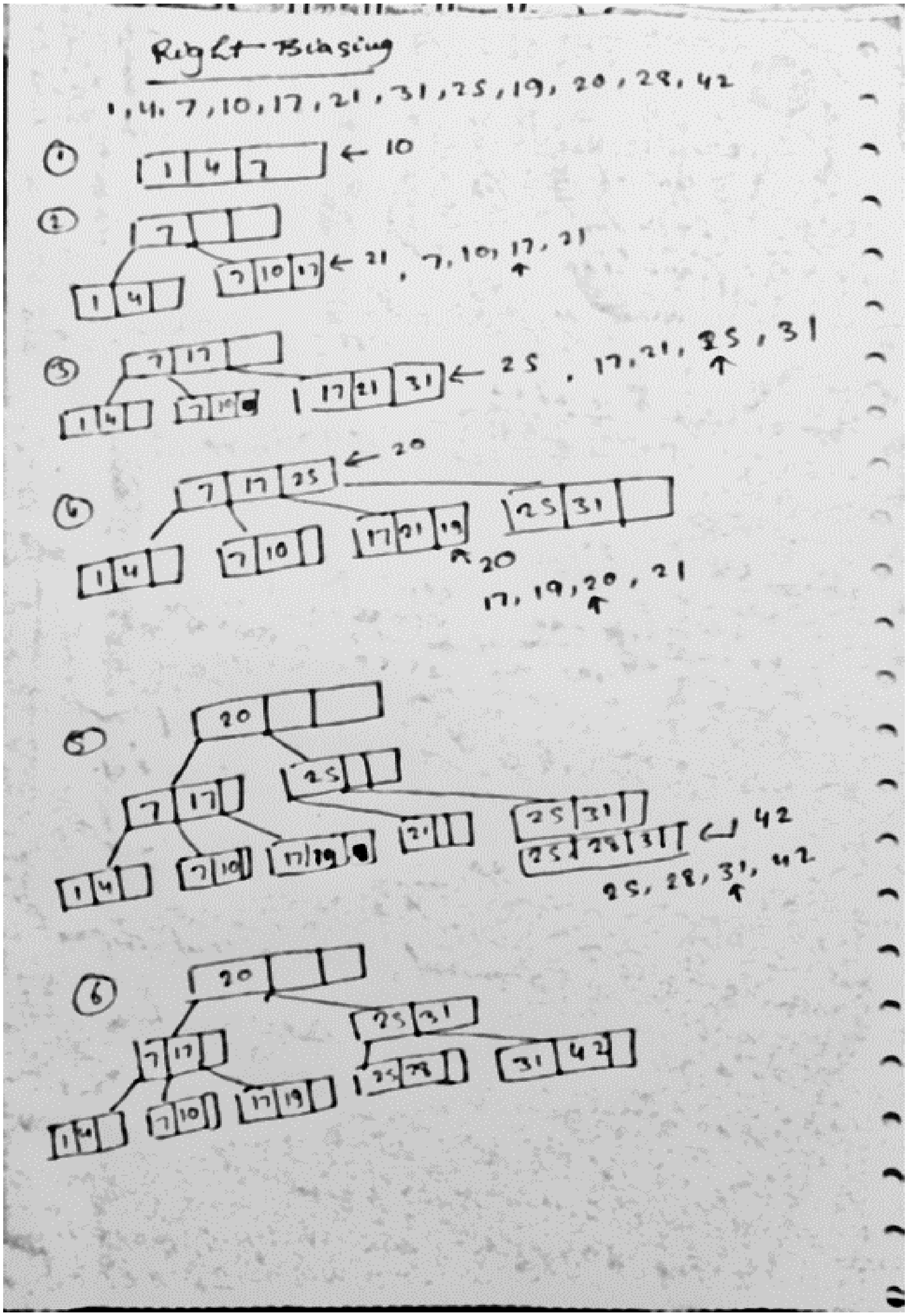
The second node contains the remaining keys i.e ’36’ and the rest of the leaf nodes remain the same.

Below is the illustration of the same:



3. Show the element wise construction of a B+ tree for the elements 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42 with order as 4. Also illustrate the deletion process by deleting any element in the root level.

Ans.



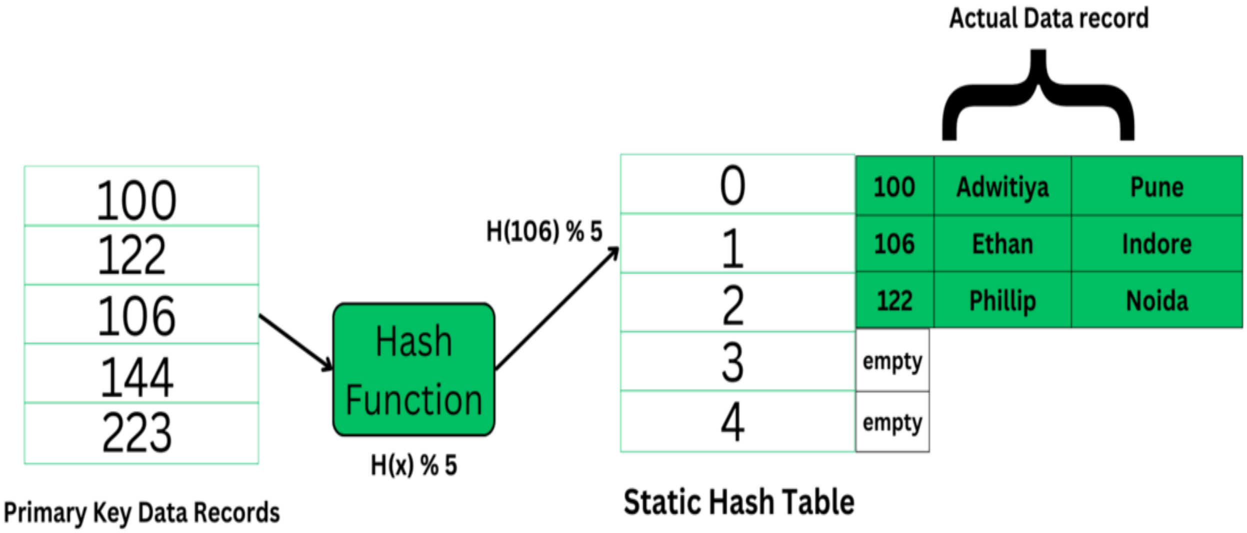
4. Describe static hashing and dynamic hashing. Provide an example for each to illustrate how they work.

Ans.

❖ STATIC HASHING:

In static hashing, the hash function always generates the same bucket's address. For example, if we have a data record for employee_id = 107, the hash function is mod-5 which is – H(x) % 5, where x = id.

Example:



➢ Static Hashing has the following Properties:

- **Data Buckets:** The number of buckets in memory remains constant. The size of the hash table is decided initially and it may also implement chaining that will allow handling some collision issues though, it's only a slight optimization and may not prove worthy if the database size keeps fluctuating.
- **Hash function:** It uses the simplest hash function to map the data records to its appropriate bucket. It is generally modulo-hash function

- **Efficient for known data size:** It's very efficient in terms when we know the data size and its distribution in the database.

❖ DYNAMIC HASHING:

Dynamic hashing is also known as extendible hashing, used to handle database that frequently changes data sets. This method offers us a way to add and remove data buckets on demand dynamically. This way as the number of data records varies, the buckets will also grow and shrink in size periodically whenever a change is made.

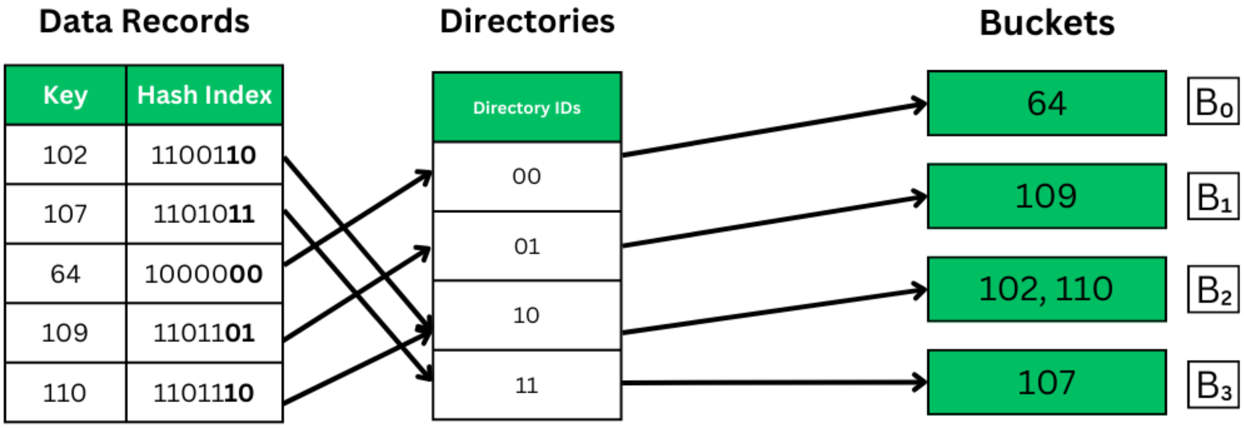
➢ Properties Dynamic Hashing:

- The buckets will vary in size dynamically periodically as changes are made offering more flexibility in making any change.
- Dynamic Hashing aids in improving overall performance by minimizing or completely preventing collision

- **It has the following major components:** Data bucket, Flexible hash function, and directories
- A flexible hash function means that it will generate more dynamic values and will keep changing periodically asserting to the requirements of the database.

Example:

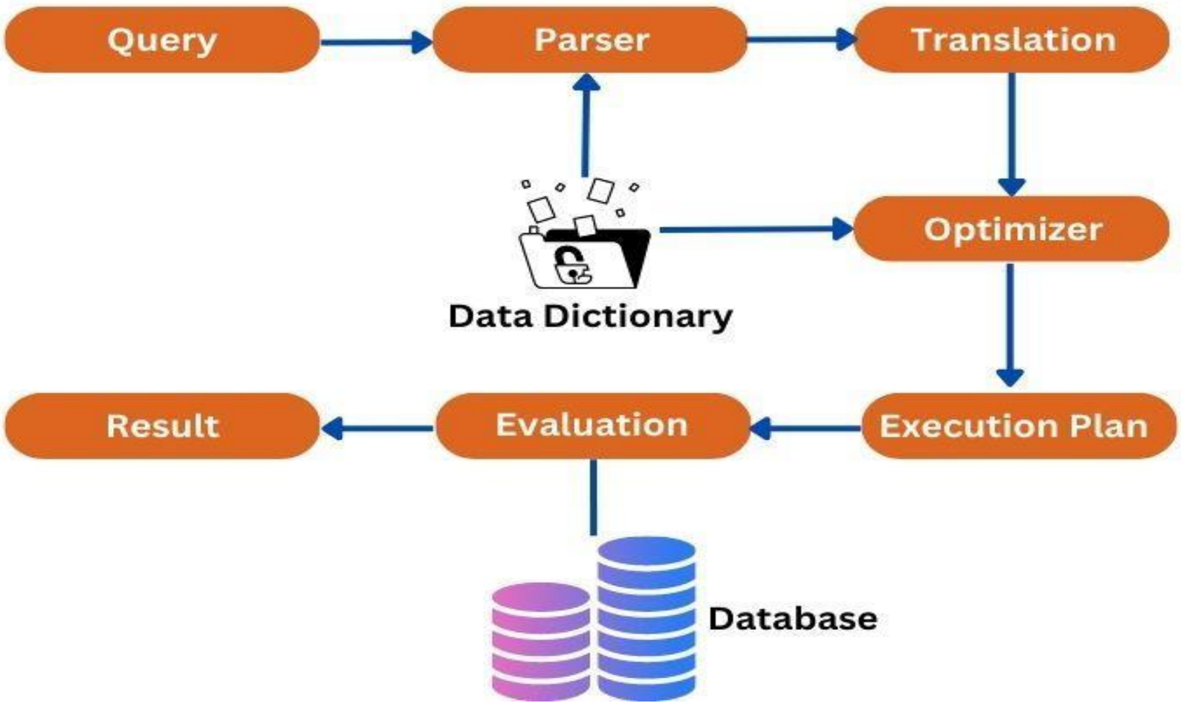
- If global depth: k = 2, the keys will be mapped accordingly to the hash index. K bits starting from LSB will be taken to map a key to the buckets. That leaves us with the following 4 possibilities: 00, 11, 10, 01.



Key	Hash Index
102	11001**10**
107	11010**11**
64	10000**00**
109	11011**01**
110	11011**10**

As we can see in the above image, the k bits from LSBs are taken in the hash index to map to their appropriate buckets through directory IDs. The hash indices point to the directories, and the k bits are taken from the directories' IDs and then mapped to the buckets. Each bucket holds the value corresponding to the IDs converted in binary.

5. (i) Sketch and concise the basic steps involved in query processing.
Ans.



- A given SQL query is translated by the query processor into a low level program called an execution plan
- An execution plan is a program in a functional language
- The physical relational algebra extends the relational algebra with primitives to search through the internal storage structure of DBMS.

Basic Steps in Query Processing

1. **Parsing and translation**
2. **Optimization**
3. **Evaluation**

1. Parsing and translation

- Translate the query into its internal form. This is then translated into relational algebra.
- Parser checks syntax, verifies relation.

2. Optimization

- SQL is a very high level language:
  - The users specify what to search for- not how the search is actually done
  - The algorithms are chosen automatically by the DBMS.
- For a given SQL query there may be many possible execution plans.
- Amongst all equivalent plans choose the one with lowest cost.
- Cost is estimated using statistical information from the database catalog.

3. Evaluation

- The query evaluation engine takes a query evaluation plan, executes that plan and returns the answer to that query.

(ii) What does query optimization involve? Describe heuristic- based query optimization thoroughly, using an example to illustrate its application

Ans.

- Query optimization is a feature of many relational database management systems and other databases such as NoSQL and graph databases. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

❖ Heuristic based query optimization:

  - Heuristics are used to reduce the number of choices that must be made in a cost-based approach.
  - Perform the SELECTION process foremost in the query. This should be the first action for any SQL table. By doing so, we can decrease the number of records required in the query, rather than using all the tables during the query.
  - Perform all the projection as soon as achievable in the query. Somewhat like a selection but this method helps in decreasing the number of columns in the query.
  - Perform the most restrictive joins and selection operations. What this means is that select only those sets of tables and/or views which will result in a relatively lesser number of records and are extremely necessary in the query. Obviously any query will execute better when tables with few records are joined.

❖ Heuristic based query optimization:

- Deconstruct the conjunctive selections into a sequence of single selection operations.

- Move the selection operations down the query tree for the earliest possible execution.

- First execute those selections and join operations which will produce smallest relations.

- Replace the cartesian product operation followed by selection operation with join operation.

- Deconstructive and move the tree down as far as possible.

- Identify those subtrees whose operations are pipelined.