Segment 6: Indexing and hashing[C201110 Nibras]

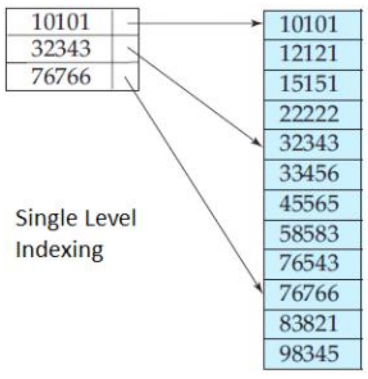
05 June 2022 19:24

☑ **Indexing:**

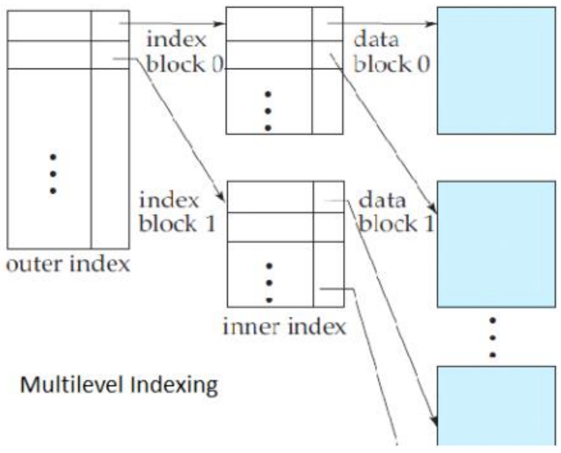
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.
- Indexing is used to optimize the performance of a database
- It minimizes the number of disk accesses required when a query is processed.
-

Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.
- Two types of Indexing are : 1) Single Level Indexing 2)Multilevel Indexing
- 

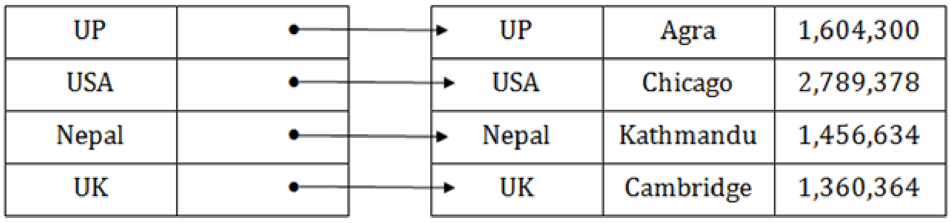
  Single Level Indexing



Multilevel Indexing

- ### 1)Single level Indexing:

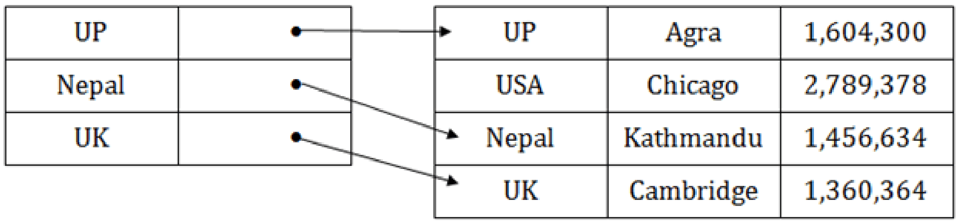
- **Ordered Indexing:** The indices which are sorted are known as ordered indices.
- **Primary Indexing:** If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.**[In Primary Indexing, primary keys are stored in sorted order]**
  - The primary index can be classified into **two types**: Dense index and Sparse index.
  - **Dense Indexing:**
    The dense index contains an index record for every search key value in the data file.



UP	•
USA	•
Nepal	•
UK	•

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

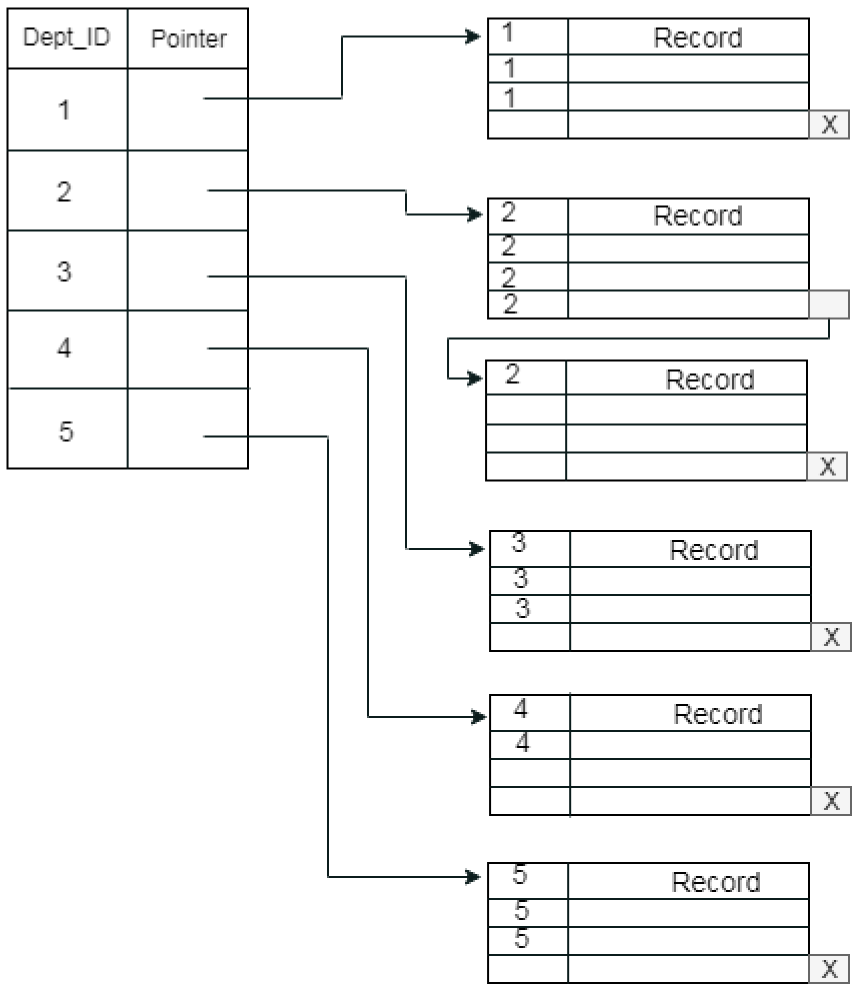
  - **Sparse Indexing:**
    In the data file, index record appears only for a few items. Each item points to a block.



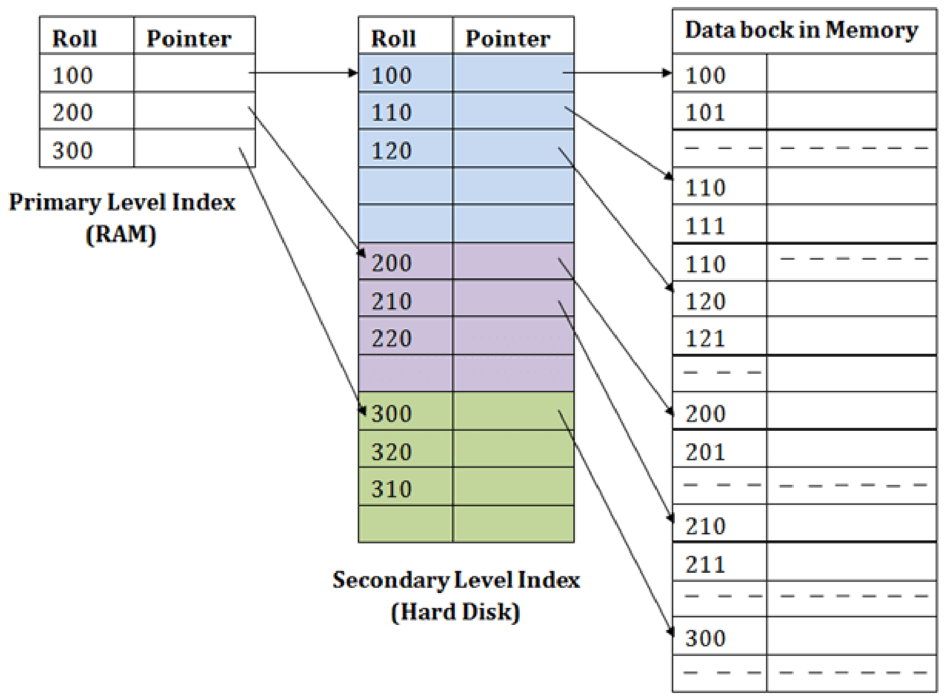
UP	•
Nepal	•
UK	•

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

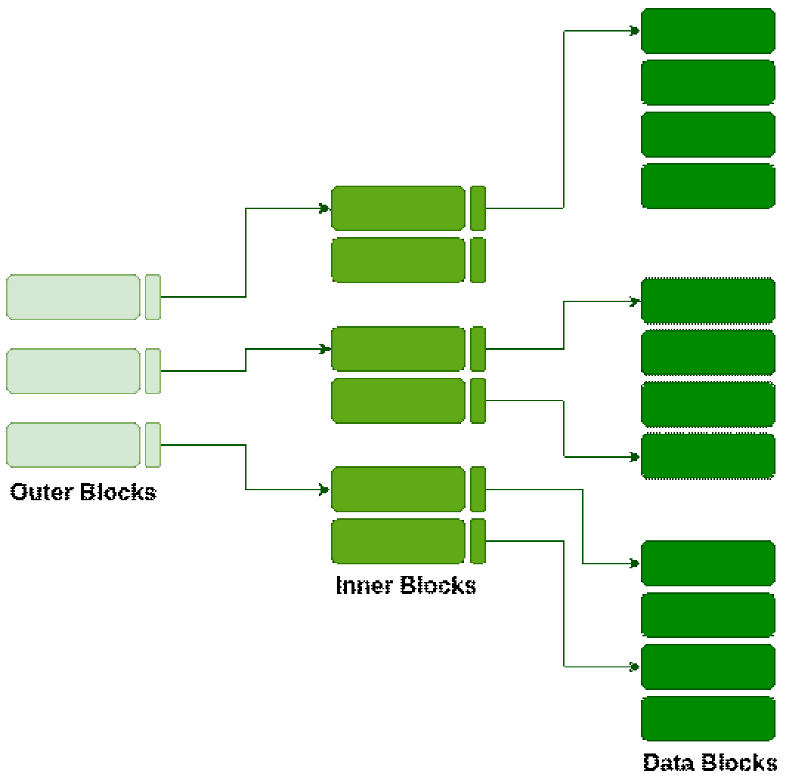
- **Cluster Indexing:** If the index is created on the basis of the non-primary key of the table But the data should be in sorted then it is known as cluster indexing.**[Non Key + Ordered]**



- <u>**Secondary Indexing:**</u> If the index is created on the basis of the <u>non-primary key</u> of the table and also <u>the data should be in Unordered</u> then it is known as secondery indexing.**[Non Key / candidate key + Unordered]**
  - **We Use this to reduce the size of mapping.**
  - In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).

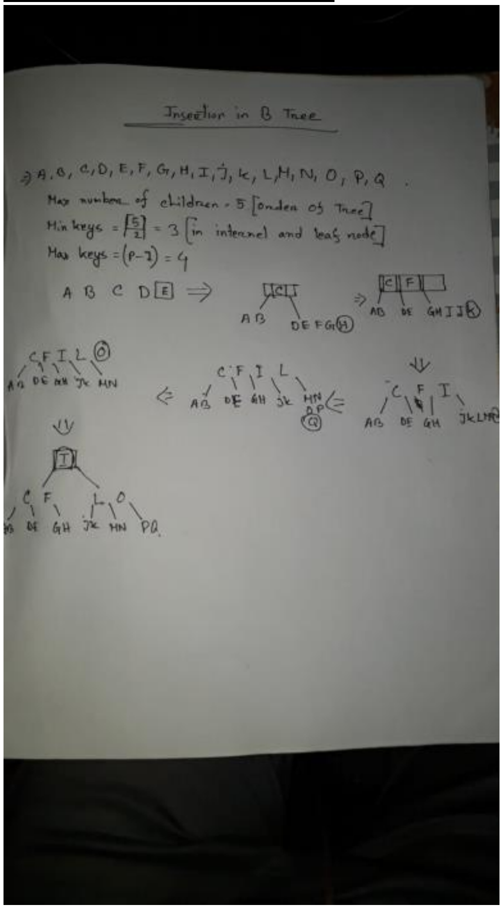


- **2)Multilevel Indexing:**



  - Multilevel Indexing in Database is created when a primary index does not fit in memory.
    With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory

with fewer overheads.

- Two types of multi-value indexing are: B tree, B+ tree:
  - **<u>B Tree Properties:</u>**
    - **If any node have, key=n ,children=n+1.**
    - **p=max children of a node=order of tree.**
    - **Root children : min=2 max=p=key+1**
    - **Internal node children: min=ceil(p/2) max=p**
    - **Internal node keys: min=ceil(p/2)-1 max=p-1**
    - **Leaf node keys: min=ceil(p/2)-1 max=p-1**
    - **Leaf node all should be in same level**
  - **Overflow: order of tree(p) > key-1**
  - **Underflow: order of tree(p) < ceil(p/2)-1**
  - **<u>Insertion In B tree:</u>**
  - 

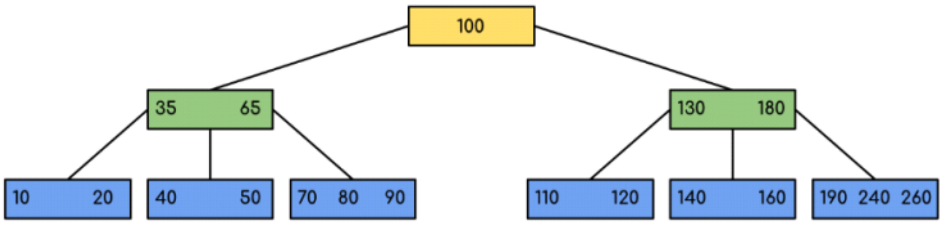
Insertion in B Tree

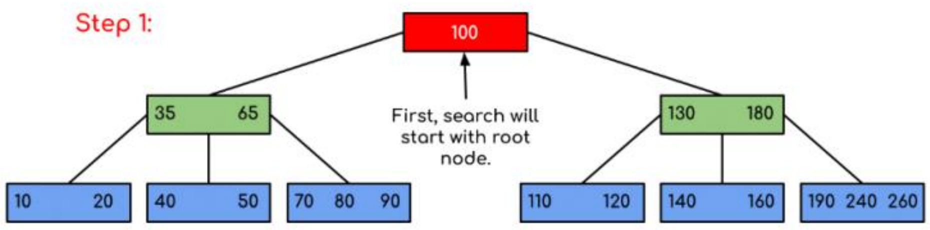
⇒ A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q.

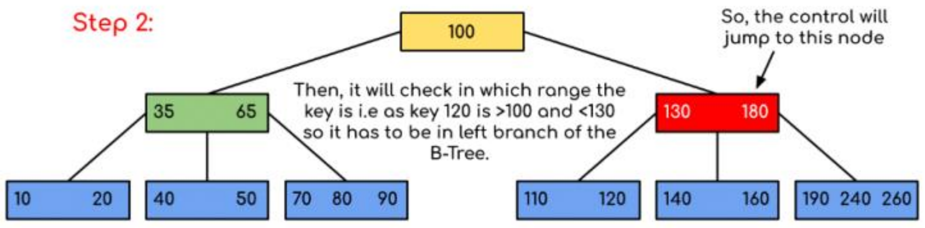
Max number of children = 5 [order of Tree]

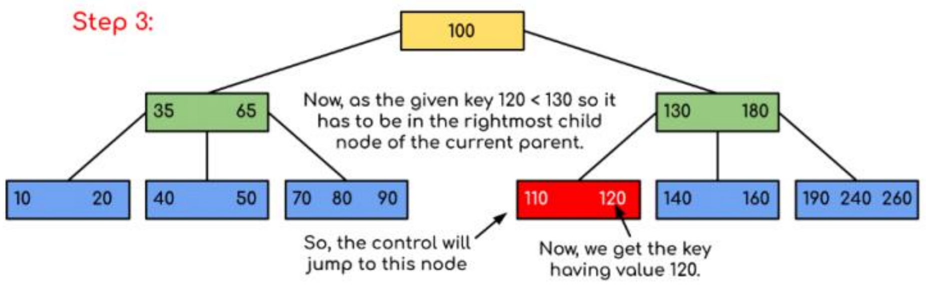
Min keys = $\lceil \frac{5}{2} \rceil$ = 3 [in internal and leaf node]

Max keys = (p-1) = 4

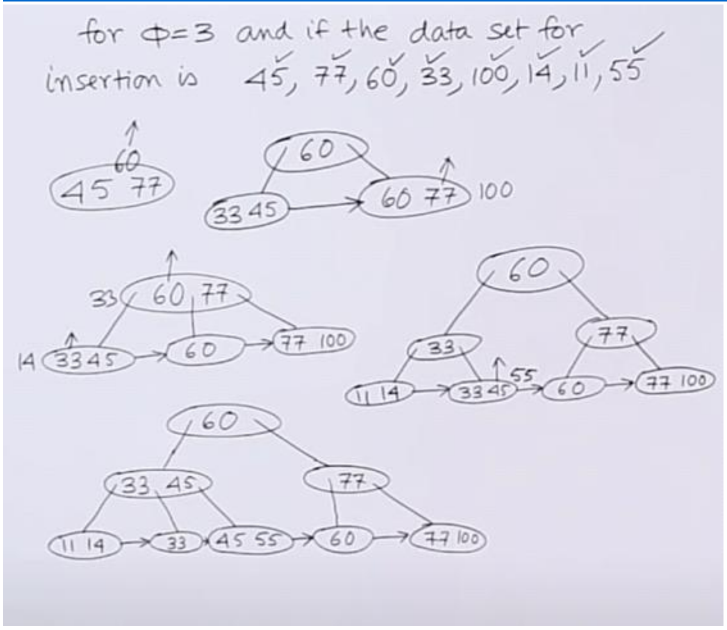
  - **<u>Searching:</u>**
    - **We will traverse from left to right until search_key <= current_key**
    - 

- 

- 
Step 2:
100
So, the control will jump to this node
Then, it will check in which range the key is i.e as key 120 is >100 and <130 so it has to be in left branch of the B-Tree.
35 65
130 180
10 20
40 50
70 80 90
110 120
140 160
190 240 260

- 
Step 3:
100
35 65
Now, as the given key 120 < 130 so it has to be in the rightmost child node of the current parent.
130 180
10 20
40 50
70 80 90
110 120
140 160
190 240 260
So, the control will jump to this node
Now, we get the key having value 120.

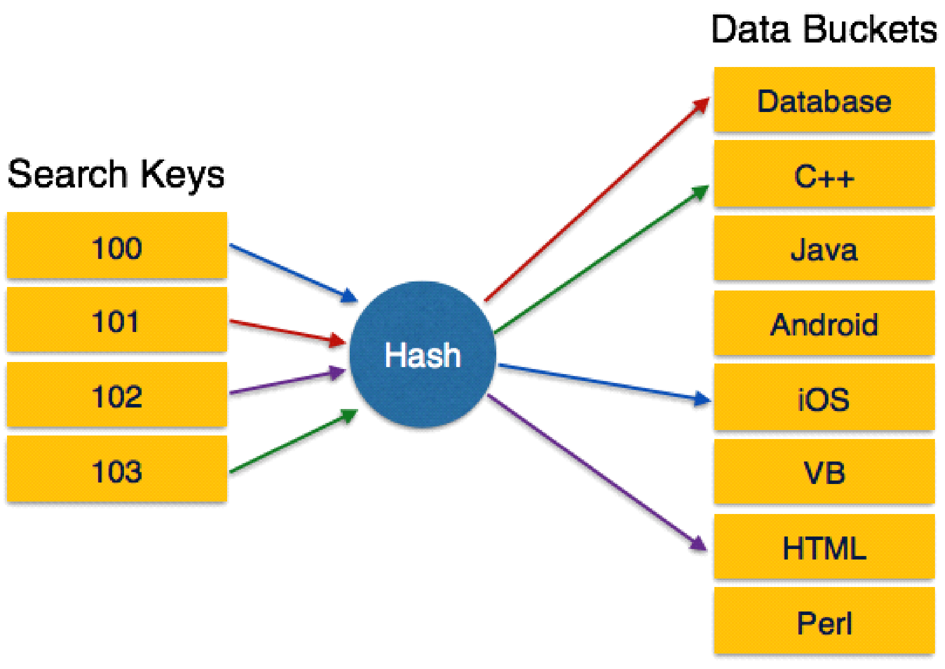
- **<u>Deleting:</u>**
  - **Follow this website:** https://www.programiz.com/dsa/deletion-from-a-b-tree
- **<u>CALCULATION:</u>**
  - **GIVEN: key, k =12B, Block size=512 B,pointer of block,pb= 10B, record of pointer Pr=8B**
    - **Equation: N*pb+(n-1)(k+pr) <=512**
      [n=number of childern]
      - **n<=17**
      - **Means maximum 17 children can have in node**
      - **Means order of tree is 17**
- **<u>B+ Trees properties:</u>**
  - **In B+ trees only leaf nodes contains pointer. Internal node contains only key.**
  - **Leaf nodes are linked using linked list**
  - **Data record only stored in leaf nodes**
  - **Each node except root can have a maximum of m children and at least m/2 children.**
  - **Each node can contain a maximum of m - 1 keys and a minimum of ⌈m/2⌉ - 1 keys.**

- **Insertion:**
- https://www.programiz.com/dsa/insertion-on-a-b-plus-tree
- 
for Φ=3 and if the data set for
insertion is 45, 77, 60, 33, 100, 14, 11, 55
- **Deletion:**
  - https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

Hashing:

- Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure in 0(1) time.
- **Hash properties:**
  - **Bucket** –Data buckets are memory locations where the records are stored. It is also known as Unit of Storage..
  - **Hash Function** – A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.
  - **Key:** A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). This allows you to find the relationship between two tables
- **Static Hashing**

  In static hashing, when a search-key value is provided, the hash function always computes the same address.



Operation

- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.

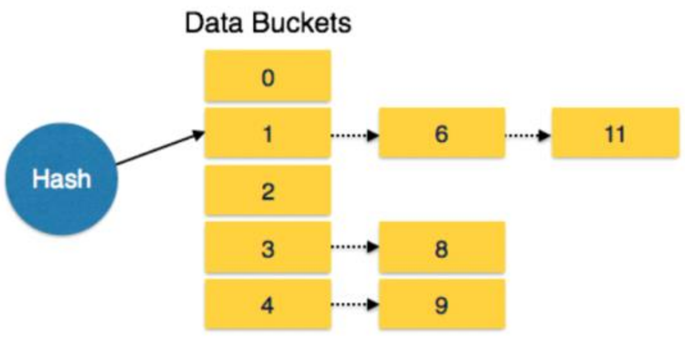
  Bucket address = h(K)

- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- **Delete** − This is simply a search followed by a deletion operation.

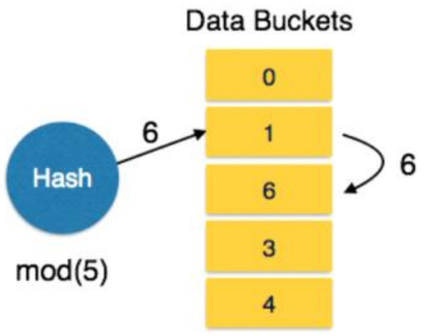
Bucket Overflow

The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.



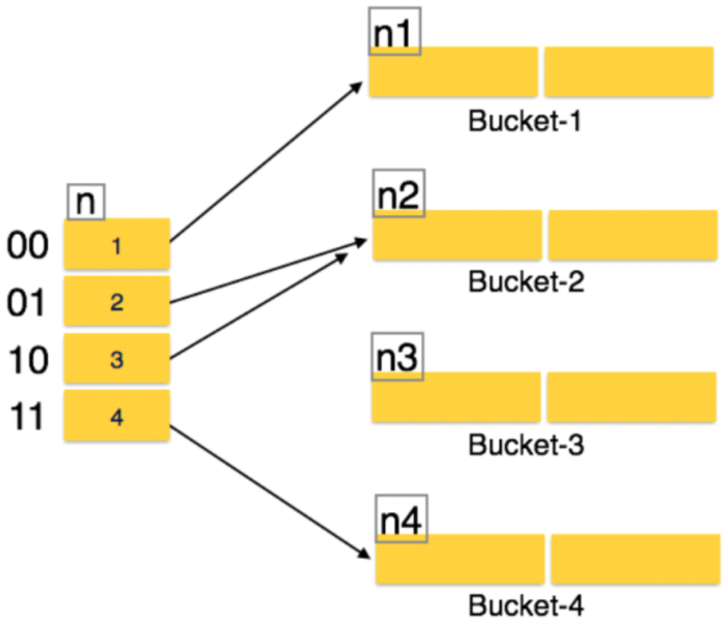
- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.



Dynamic Hashing

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



120<=100-go to right.