Parul® University

Database Management System-203105251

Kunj Thakor, Assistant Professor

Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

Topics

Storage strategies:

- Indices
- B-trees
- hashing.

Parul®
University

DIGITAL LEARNING CONTENT

Indices

Parul®
University

DIGITAL LEARNING CONTENT

What is Database Indexes?

- Indexes are special lookup tables that can be used by the database search engine to speed up the retrieval of results.
- A database index is a data structure that increases the speed of operations on a database table for data retrieval.
- An index is somewhat similar to the index on the back of a book in a database.
- Indexes are used to very easily extract data from the database.
- The indexes will not be used by the users, they are simply used to speed up searches / queries.
- It takes more time to update a table with indexes than to
- update a table without (because the indexes require an update as well.

- **Syntax:**

CREATE INDEX index_name
ON table_name (column1, column2, ...);

For Example:

Example:

```
CREATE INDEX idx_studentaddress
ON student (studentaddress);
```

- Indexing is a means of maximising a database 's output by minimising the amount of disc access needed when a query is processed.
- It is a type of data structure which is used to find and navigate the data in a database easily.

Parul®
University

DIGITAL LEARNING CONTENT

The structure of Index in database.

A few database columns are used to construct indexes.

Search Key	Pointer

- The first column is the Search key that includes a replica of the table 's primary or candidate key. In order for the related data to be retrieved easily, these values are stored in sorted order.
- The second column is the Storage Reference or Pointer, which includes a series of pointers containing the disc block address where you can locate the particular key value.

Attributes Of Indexing

- **Access Types:** This extends to access types such as value-based searching, range access, etc.
- **Access Time:** This refers to the time taken to locate a single element of data or a group of elements.
- **Period of insertion:** refers to the time taken to locate the correct space and upload new data.
- **Deletion Time:** Time needed for identifying and removing an object and updating the index structure.
- **Space Overhead:** This applies to the extra space needed by the index.

Parul® University

DIGITAL LEARNING CONTENT

Different Indexing Methods

- Parse Indexing
- Dense Indexing

Primary Indexing **PI**

Clustering Indexing **CI**

Secondary Indexing **SI**

Primary Indexing

What Is Primary Indexing?

- If the index is provided by the table 's primary key, it is referred to as the primary index.
- For each record, these primary keys are special.
- Since primary keys are stored in sorted order, the search operation 's efficiency is very effective.

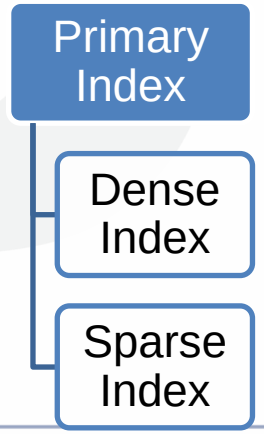
For Example

Student (enrollNo, Name, Address, City, MobileNo) ***[enrollNo is primary key]***

CREATE INDEX idx_StudentRno
ON Student (enrollNo);

- The primary index can be classified into two types:

➢ Dense index
➢ Sparse index



Dense Index

- There is an index record for each search key value in the database in a dense index.
- This makes the search easier, but needs more room for index documents to be stored.
- In the index table, the number of records is like the number of records in the main table.
- Index records contain the value of the search key and a pointer to the real disc record.

101	
102	
103	
104	

Index Table

Rno.	Name
101	Vikas
102	Meet
103	Jay
104	Mark

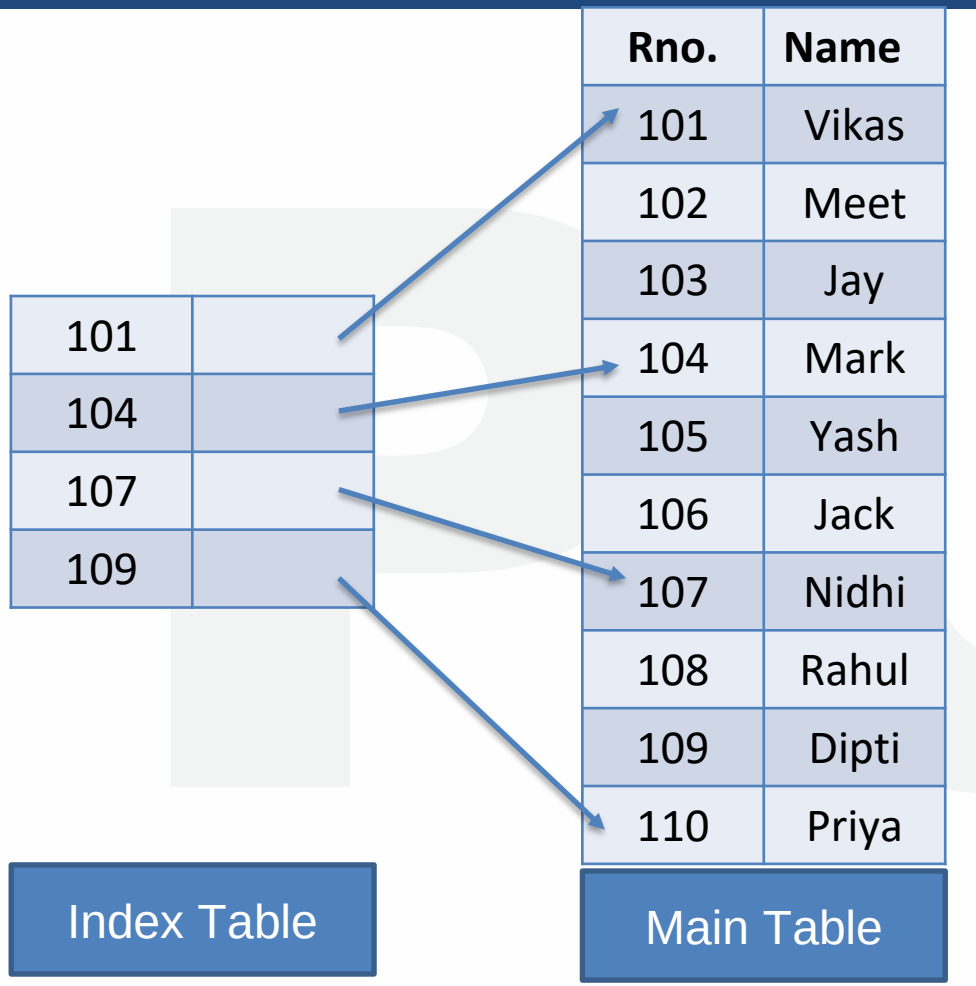
Main Table

Parul® University

Sparse Index

- Index records for any search key are not generated in the sparse index.
- The index record only appears in the data file for a few things.
- It needs less space, less overhead maintenance for insertion, and deletions, but is slower compared to the dense record locating index.
- To search for a record in a sparse database, we search for a value in the index we are searching for that is less than or equal to the value.
- Linear searches are carried out to recover the desired record after the first record is retrieved.
- In sparse indexing, the size of the index table often increases as the size of the main table expands.

Sparse Index



Secondary Indexing

- In secondary indexing, another level of indexing is added in order to reduce the mapping scale.
- The vast range for the columns is initially chosen in this process, so that the mapping scale of the first stage becomes minimal.
- Then each set is broken further into smaller sets.
- The first level mapping is located in the primary memory, so that the fetching of addresses is easier.
- The second stage mapping and real data are contained in the secondary memory (hard disc).
- If you wish to find the 112 roll record, then the highest entry in the first stage index that is smaller than or equal to 112 will be scanned. At this stage, he'll get 101.
- Then it does max (112) < = 112 again at the second index stage and gets 111. Now it goes to the data block using the address 111, and begins looking for each record before it gets 112.
- Here is how this approach conducts a quest.
- Inserting, upgrading or removing are often conducted in the same way

Secondary Indexing

101	
201	
301	
401	

Primary Index

101	
111	
201	
211	
301	
311	

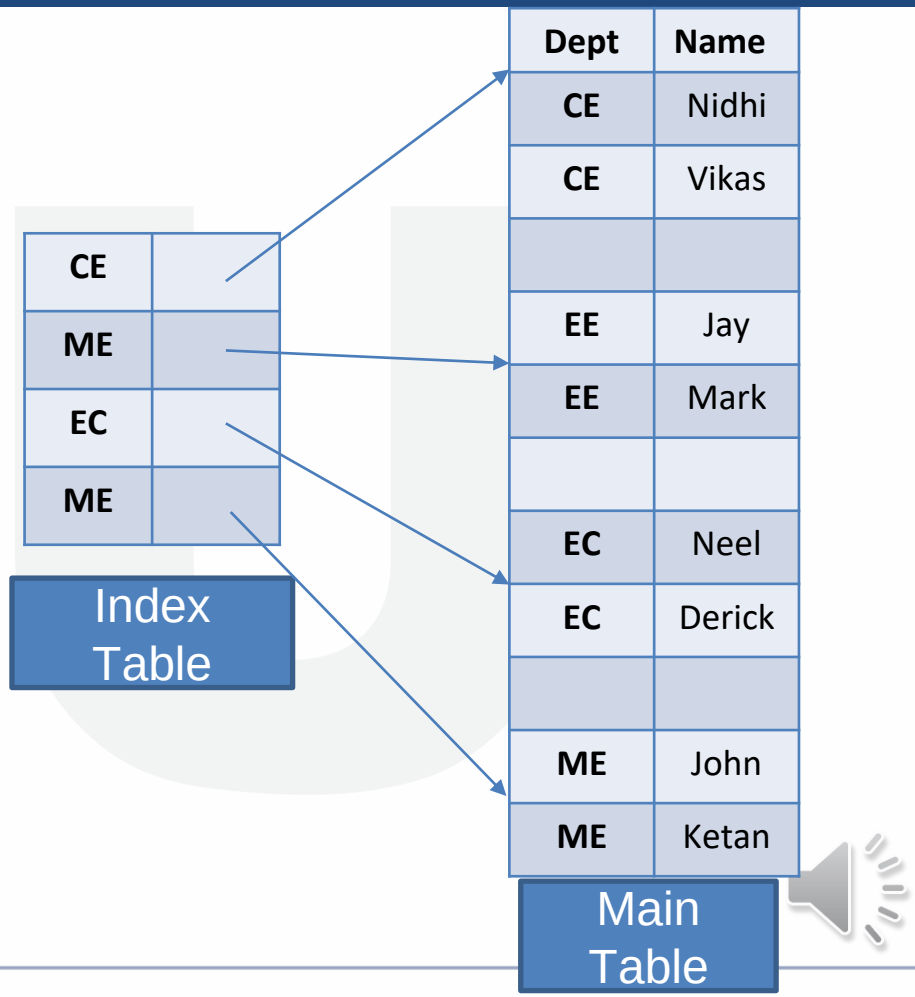
Secondary Index

Rno.	Name
101	Vikas
102	Meet
111	Mark
112	Yash
201	Nidhi
202	Rahul
211	Priya
212	Neel

Main Table

Clustering Indexing

- The index is often created for non-primary main columns that may not be unique to each record.
- In this case, we can group two or three columns to get the unique value and produce an index out of them to classify the record faster. A clustering index is called this form.
- Documents with common characteristics are grouped and indexes for these categories are developed.



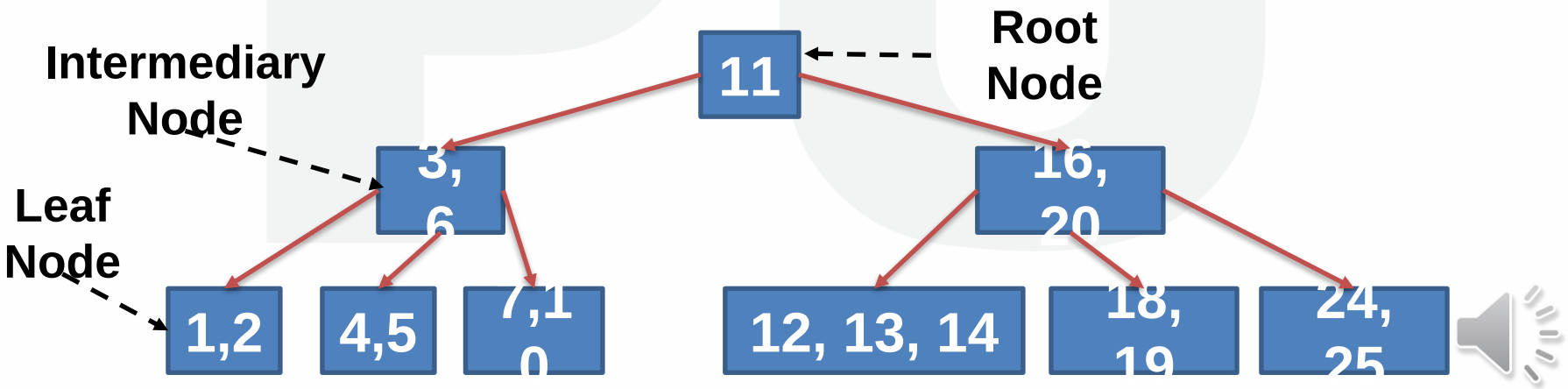
Parul®
University

DIGITAL LEARNING CONTENT

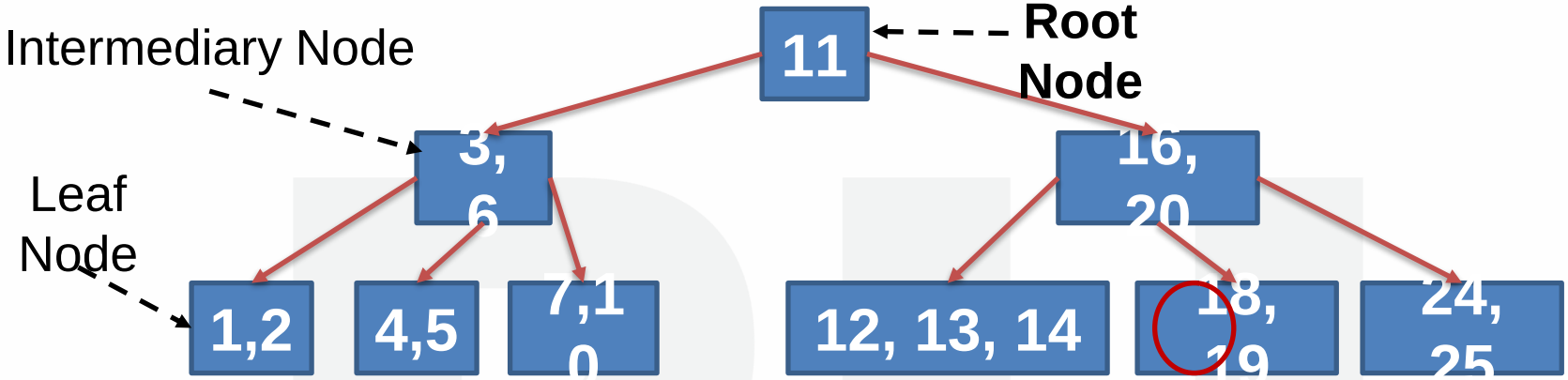
B-Tree

B-Tree

- B-tree is a data structure that store data in its node in sorted order. We can represent sample B-tree as follows.
- B-tree stores data in such a way that each node contains keys in ascending order.
- Each of these keys has two references to another two child nodes.
- The left side child node keys are less than the current keys and the right side child node keys are greater than the current keys.

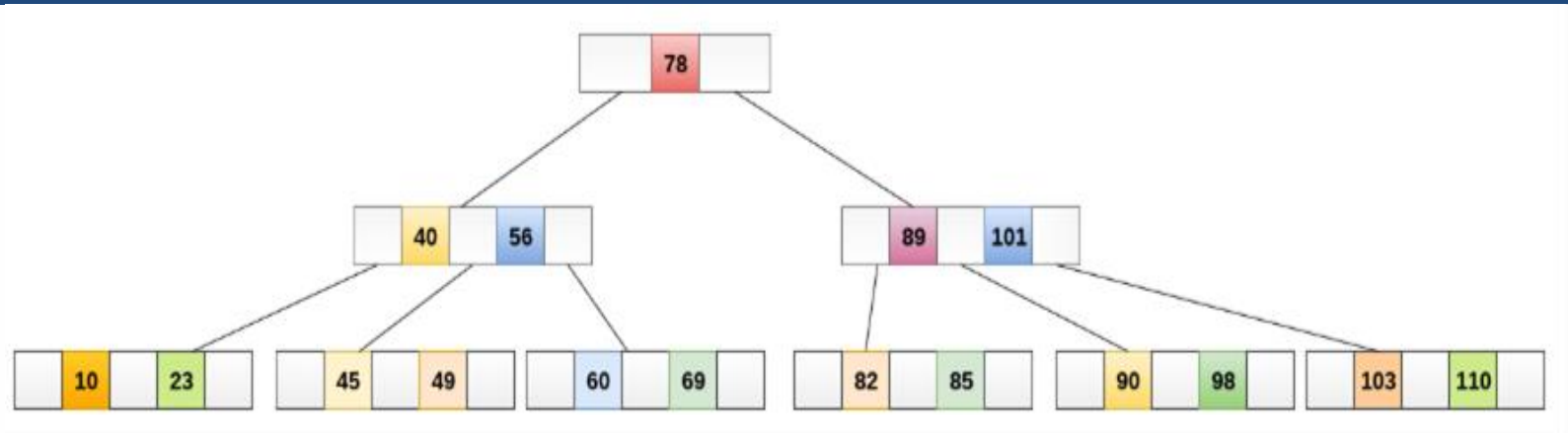


Searching a record in B-tree



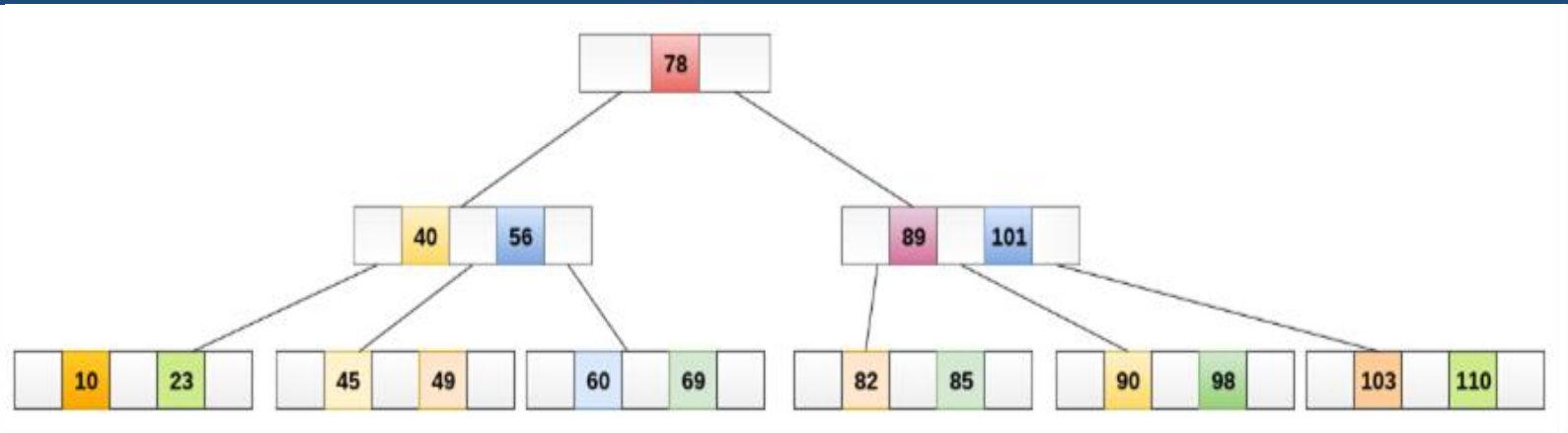
- Suppose we want to search 18 in the above B tree structure.
- First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 18.
- So, in the intermediary node, we will find a branch between 16 and 20 nodes.
- Then at the end, we will be redirected to the fifth leaf node. Here DBMS will perform a sequential search to find 18.

Searching In B-Tree



- If we search for an item 49 in the following B Tree. The process will something like following :
- Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
- Since, 40<49<56, traverse right sub-tree of 40.
- 49>45, move to right. Compare 49.
- match found, return.
- **Searching in a B tree depends upon the height of the tree. The search algorithm**

Searching In B-Tree



- If we search for an item 49 in the following B Tree. The process will something like following :
- Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
- Since, 40<49<56, traverse right sub-tree of 40.
- 49>45, move to right. Compare 49.
- match found, return.
- **Searching in a B tree depends upon the height of the tree. The search algorithm**

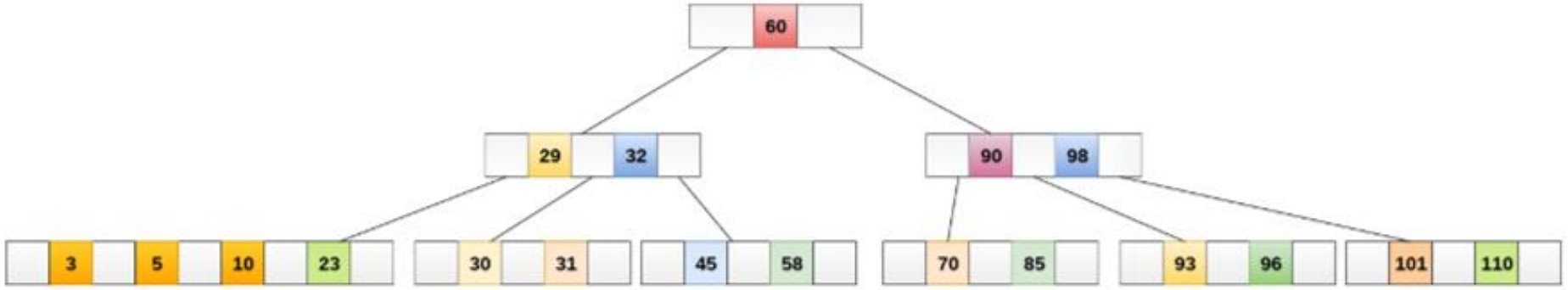
Parul® University

Inserting a record In B-Tree

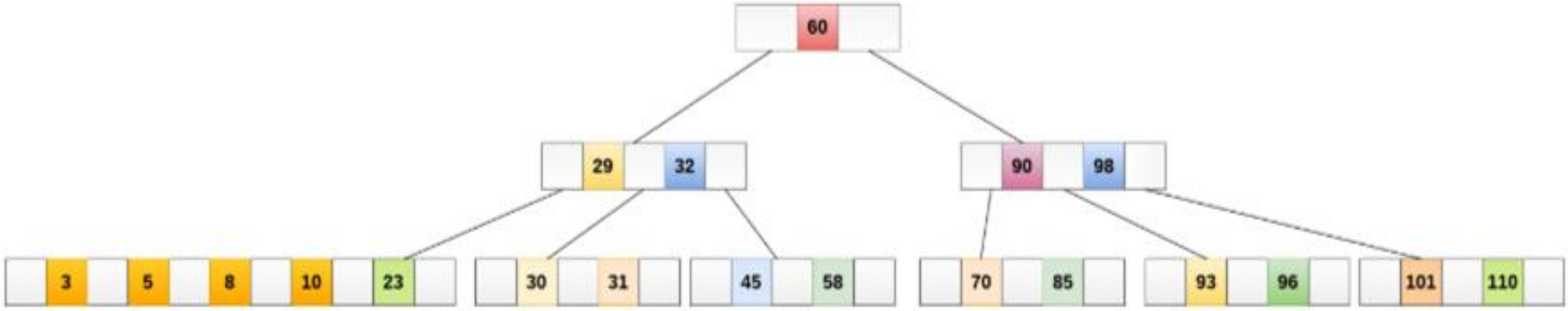
- Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.
- Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
- If the leaf node contain less than m-1 keys then insert the element in the increasing order.
- Else, if the leaf node contains m-1 keys, then follow the following steps.
  - Insert the new element in the increasing order of elements.
  - Split the node into the two nodes at the median.
  - Push the median element upto its parent node.
  - If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Inserting a record In B-Tree

- Insert the node 8 into the B Tree of order 5 shown in the following image

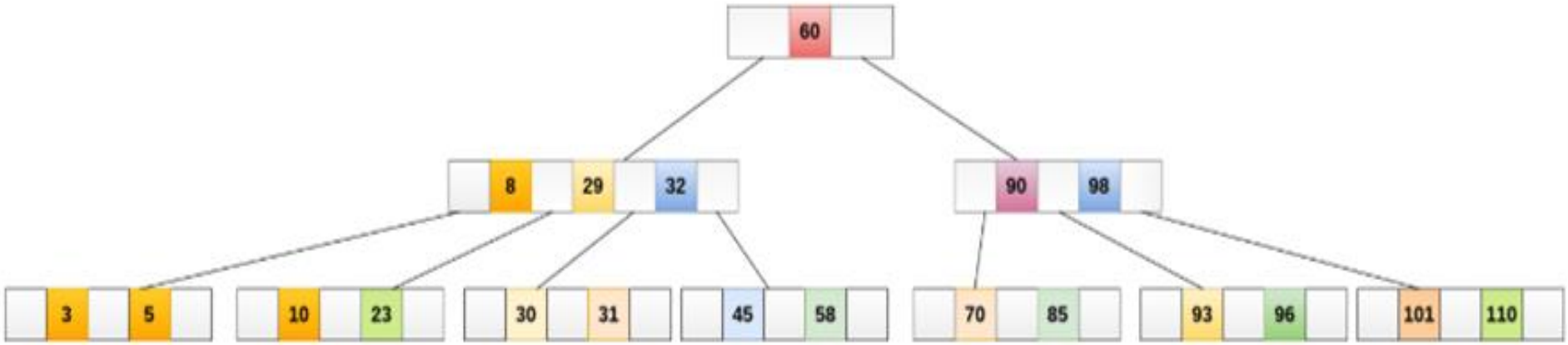


- 8 will be inserted to the right of 5, therefore insert 8.



Inserting a record In B-Tree

- The node, now contain 5 keys which is greater than (5 -1 = 4) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.



Parul® University

Advantages & Disadvantages Of B-Tree Indices

- **Advantages of B-Tree indices:**
  - May use less tree nodes than a corresponding B^+-Tree.
  - Sometimes possible to find search-key value before reaching leaf node.
- **Disadvantages of B-Tree indices:**
  - Only small fraction of all search-key values are found early
  - Non-leaf nodes are larger, so fan-out is reduced. Thus, B-Trees typically have greater depth than corresponding B^+-Tree
  - Insertion and deletion more complicated than in B^+-Trees
  - Implementation is harder than B^+-Trees.

Parul®
University

DIGITAL LEARNING CONTENT

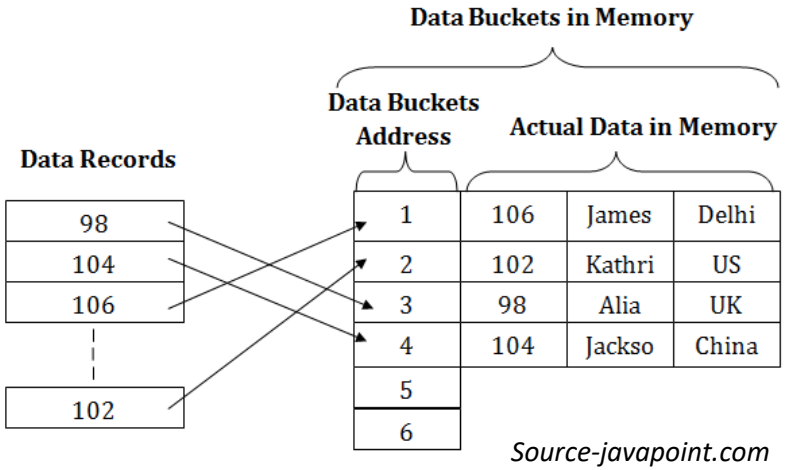
Hashing

Hashing

- For a huge database, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data.
- Hashing is a technique to directly search the location of desired data on the disk without using index structure.
- Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored known as a data block or data bucket.
- Hashing uses hash functions with search keys as parameters to generate the address of a data record.
- Data bucket: Data buckets are the **memory locations where the records are stored**.
- Hash Function: Hash function is a **mapping function that maps all the set of search keys to actual record address**. Generally, hash function uses primary key to generate the hash index – address of the data block.

Static Hashing

- In the static hashing, **the resultant data bucket address will always remain the same**.
- Therefore, if you generate an address for say Student_ID = 10 using hashing function mod(3), the resultant bucket address will always be 1. So, you will not see any change in the bucket address.
- Therefore, in this static hashing method, the number of data buckets in memory always remains constant.



Source-javapoint.com

Operation In Static Hashing

➢ **Searching a record**

- When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

➢ **Insert a Record**

- When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

➢ **Delete a Record**

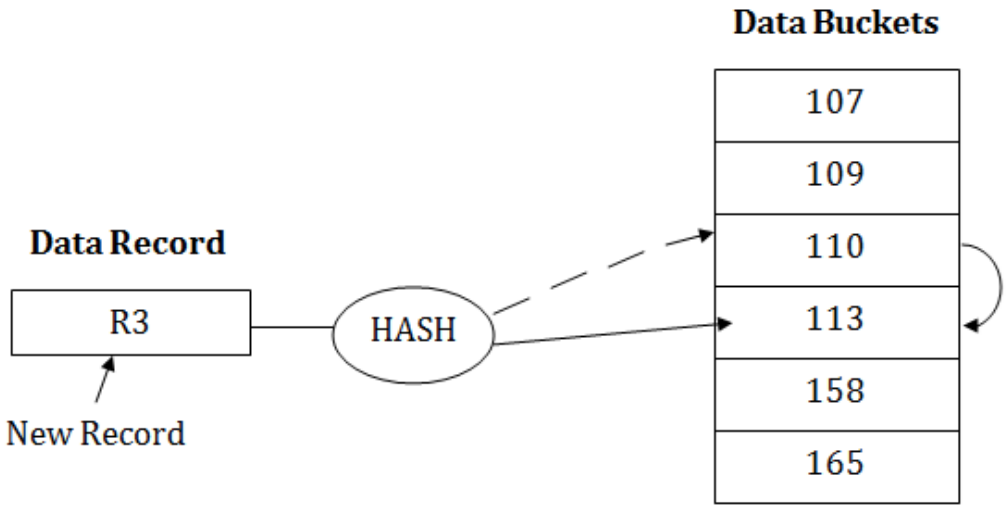
- To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

➢ **Update a Record**

- To update a record, we will first search it using a hash function, and then the data record is updated.
- 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. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

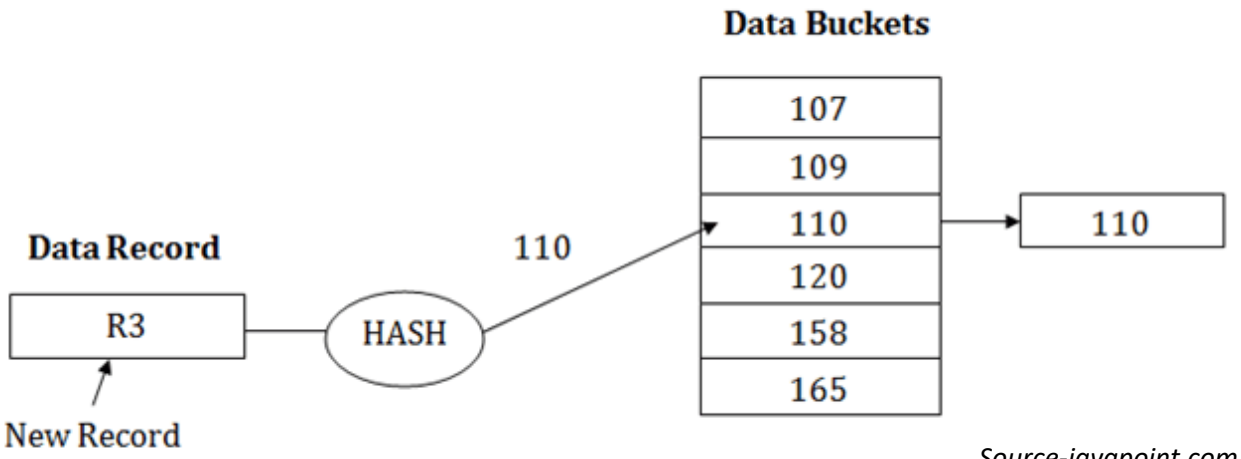
Open Hashing

- When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.
- **For example:** suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



Close Hashing

- When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.
- **For example:** Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Source-javapoint.com

Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, 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.
- **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.

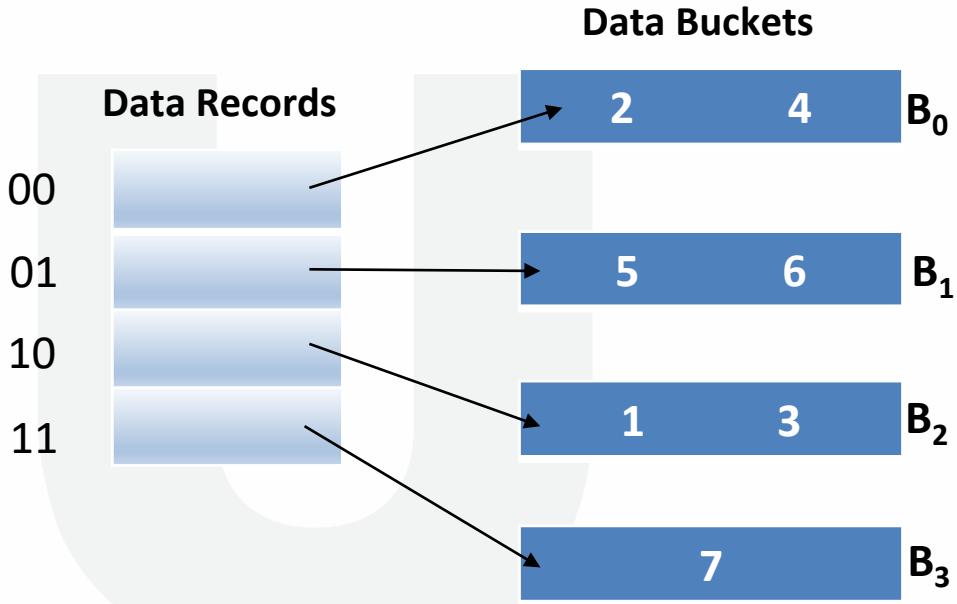
Dynamic Hashing

- **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.

For Example

- Consider the following grouping of keys into buckets, depending on the prefix of their hash address

Key	Hash Address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111



- The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.

Advantages & Disadvantages

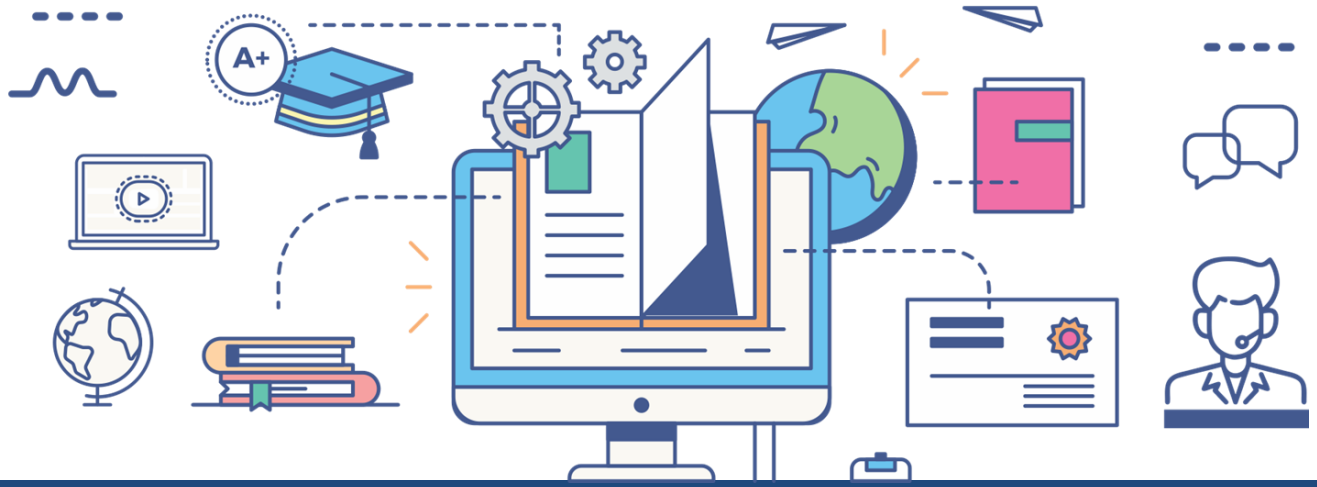
➢ **Advantages of dynamic hashing**

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

➢ **Disadvantages of dynamic hashing**

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in