MODULE IV

Index Sequential Access & Prefix B+ Tree

Indexed sequential file structure provides a choice between two alternative views of a file

- Indexed – The file can be seen as a set of records that are indexed by keys.
- Sequential- The file can be accessed sequentially (physically contiguous records-no seeking) returning records in order by key.

Eg – suppose in an employee database, an employee has to be searched – here indexed access is required and sometime all the employees must be accessed – here sequential access is required.

Definition of Indexed sequential access:

Indexed sequential access is the ability to access the record both sequentially and randomly (indexed access) on the same records, which are ordered by key. Here the access is not a single access ie all records are not continuous.

Maintaining a Sequence set:

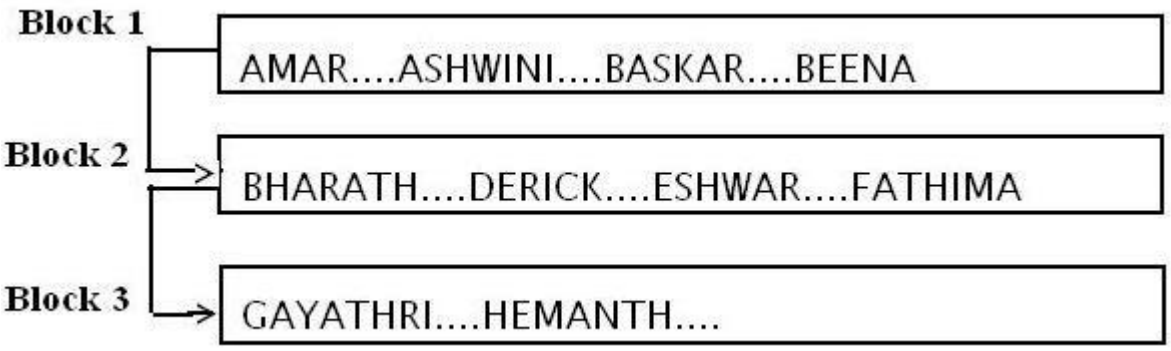
A sequence set is a set of records in sequence (order). The order is maintained even if new records are added or records are deleted.

The use of blocks:

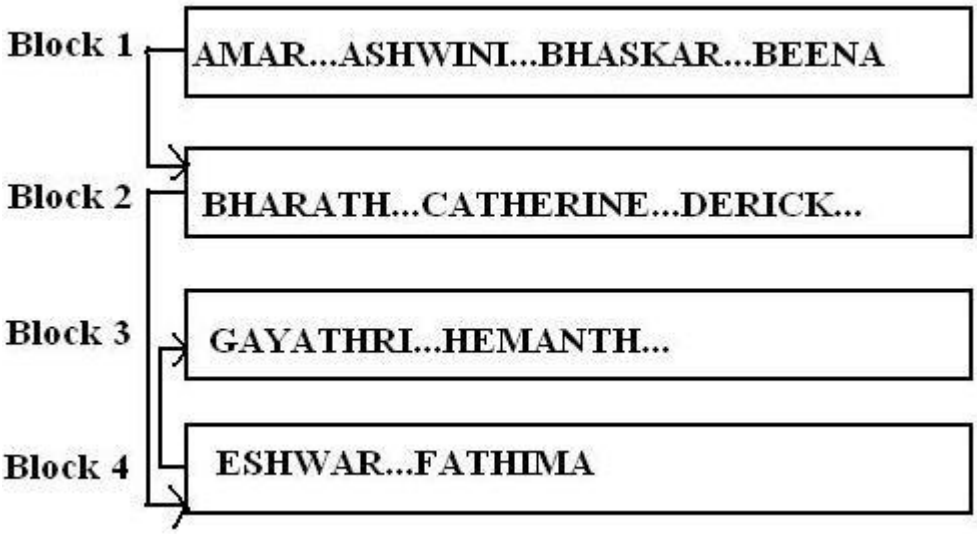
The sorting of an entire file after addition or deletion is an expensive process. Instead blocks are used to localize the changes.

A block consists of few complete records and link fields connecting to the preceding blocks. The consecutive blocks may not be adjacent.

The entire block is read at ones or written at ones. The insertion of new records to a full block, causes an overflow of records. This is handled by block splitting.



Suppose a record with key 'CATHERINE' is added, that should be added in block2 (according to the order), so the node is splitted.



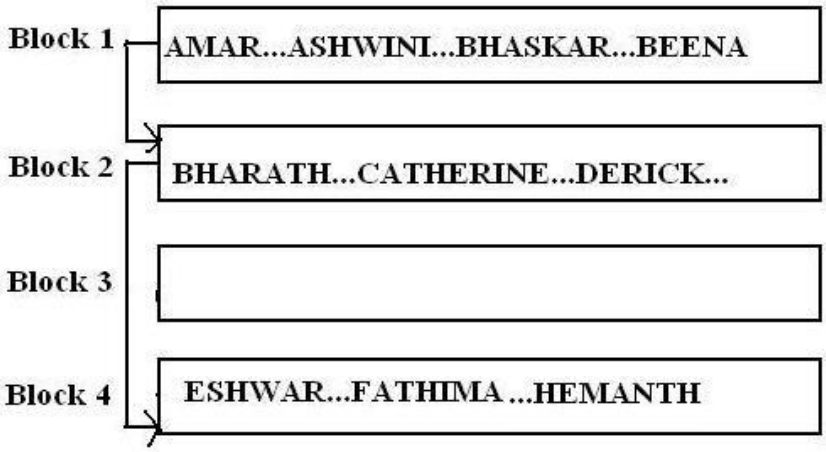
The 2nd half of block 2 is found at block 4 and the link fields are linked in order.

Deletion of records, if a block is less than half full, then it is an underflow of block.

When an underflow occurs, there are 2 solutions –

- If a neighboring node is also half full, we can merge the two nodes, freeing p one block for reuse.
- If the neighboring nodes are more than half full, redistribution of records takes place between the blocks.

Suppose 'Gayatri' record from above diagram is deleted. Then, block 3 and block 4 records are merged into block 4 and block 3 can be reused.



Disadvantages of using blocks:

- Blocked file takes up more space than the unblocked file because of internal fragmentation within a block.
- The order of records is not necessarily physically sequential(actually sequential in memory) throughout the file. The maximum guaranteed extent of physical sequentiality is within a block.

Choice of block size:

A block contains records arranged sequentially. In one access, we can access a block of records. So a larger block size is always required. But there will be large internal fragmentation. So there are two considerations to keep in mind when choosing a block size.

Consideration 1: The block size should be such that we can hold several blocks in memory at once. Eg: In performing a block split or merging, we want to hold at least two blocks in memory at a time.

Consideration 2: Reading in or writing out a block should not take very long. Even if we had an unlimited amount of memory, we would want to place an upper limit on the block size, so that we need not read entire file to get a single record.

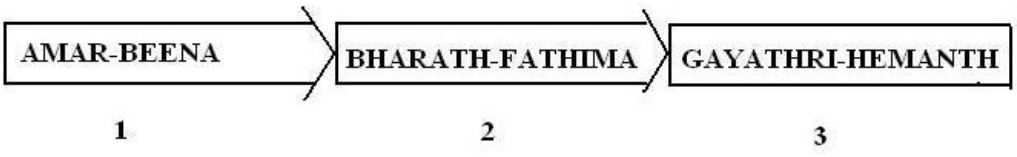
Consideration 2(Redefined):

The block size should be such that we can access a block without having to bear the cost of a disk seeks within the block read or block write operation.

Adding a simple index to the sequence set:

The records are maintained in blocks and the records are sequential. But if random access of records is required an index file is maintained.

A simple, single level index for these blocks is created. The index file consists of a key (of the last record) and the block number. For Ex:



The index file is built as follows

Key	Block number

BEENA 1

FATHIMA 2

HEMANTH 3

Thus the key of last record and block number helps in accessing the records randomly.

-The combination of index and sequence set of blocks provides a complete indexed sequential access.

-If we need to access a specific record, we check the index and then retrieve the correct block.

-If we need sequential access, we start at the first block and read through the linked list of blocks until we have read them all.

The entire index file must be in memory because

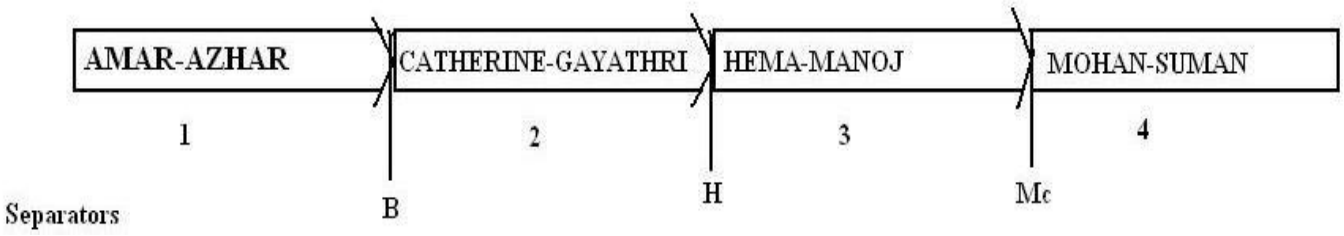
-To find the specific record, we use the binary search on the index file, which takes too many seeks if the file is stored in the secondary storage device.

-As the blocks in the sequence set are changed (by spitting, merging, redistribution), the index also has to be updated.

The contents of the index: separators instead of keys

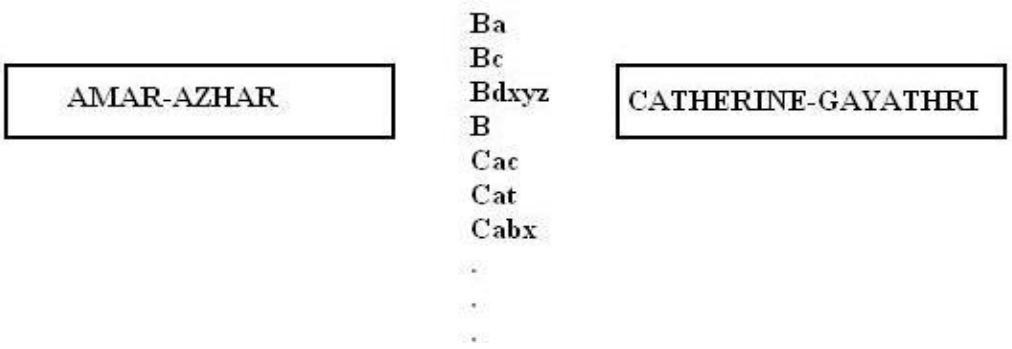
Index is built to assist us in selecting the right block to read a particular record. The key of a record is used to build an index in B-trees. But instead of using a key any simple separator can be used, to separate the two blocks.

Eg: If the blocks are as shown below



Separators can be a single character or group of characters. The 1st separator is B, ie all keys less than B are in block 1 and all keys greater than B are in block2. Similarly 'Mc' is the separator between block 3 and block 4.

The separator that can be used between block a and block 2 are



The shortest separator 'B' is used in the example as it saves memory in index file. The decision to retrieve the block to the right of the separator or to the left is done according to the following rule:

Relation of key and separators	Decision
Key<Separator	Go Left
Key=Separator	Go Right
Key>Separator	Go Right

C++ function to find the shortest separator:

```
Void FindSeparator ( char *k1, char *k2,char *sep)
{
        //k1,k2← variables representing 2 possible separators.
        //sep← Actual separator.
        While (1)
        {
                *sep=*k2; sep++;
                If(*k2!=*k1) break;// Stop when difference is found.
                If( *k2==0) break;
                K1++;k2++;//Move to the next character of keys.
        }
        *sep=0;//Null terminate the separator string.
}
```

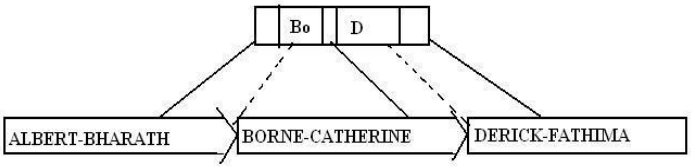
The Simple Prefix B+Tree

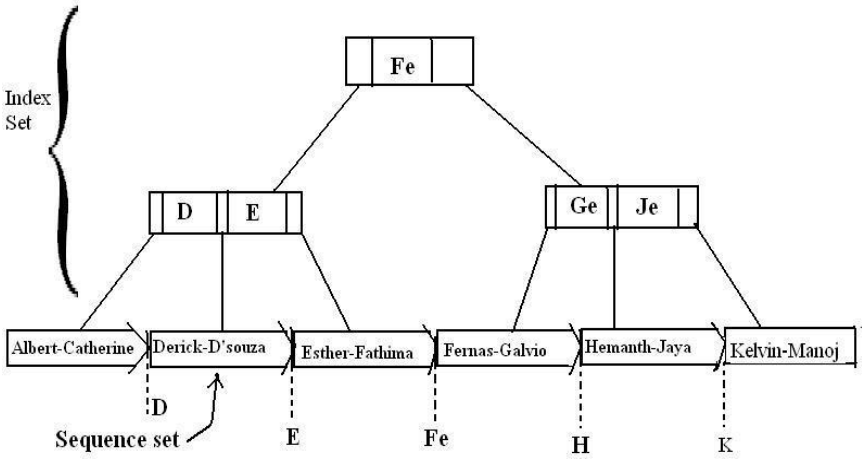
The BTrees index is called the index set. The sequence set and index set, taken together forms a file structure called a Simple Prefix B+Tree.

The modifier simple indicates that the index set contains the shortest separator, and 'prefix' means that the prefixes of the keys are used rather than the whole key.

The separators selected may be of various size

Eg- Bo, D,Ea etc





Maintenance of simple prefix B+Tree:

Maintenance occurs when insertion or deletion of records occur. There are two possible chances-

- Changes localized to single blocks in the sequence set.
- Changes involving multiple blocks in sequence set.

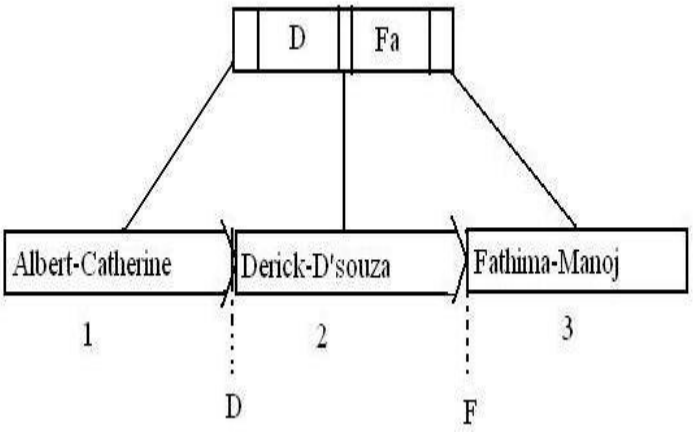
1) Changes localized to single blocks in sequence set:

The changes in the sequence set due to insertion or deletion, if these process does not result in splitting, merging or distribution, then it does not changes any other 'block or the index set'.

-The effect of deletions or insertions on sequence set is limited to within particular block(in which changes occurred).

-Since other blocks does not change, and since no records are moved between the blocks, there is no change in the index set.

Eg:



If a record with the key 'Bharath' is added to block1, and block 1 does not cause an overflow, then there is no splitting of block 1, not the index set changes.

2) Changes involving multiple blocks in the sequence set

If insertion or deletion causes changes in the other blocks, due to splitting, merging or reduction, then it also changes the index set. Thus multiple blocks are affected.

-If splitting, merging or reduction is necessary, perform the operations just as you would do if there is no index set and then update the changes in the index set.

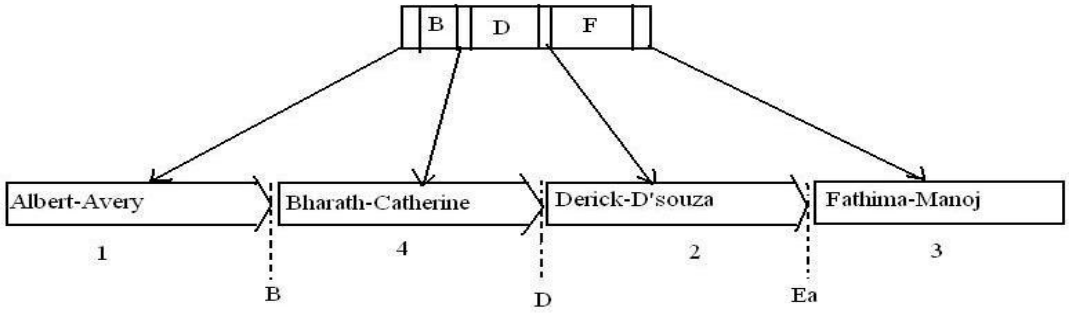
-If blocks are split in the sequence set, a new separator must be inserted into the index set.

-if blocks are merged in sequence set, a separator is removed from the index set.

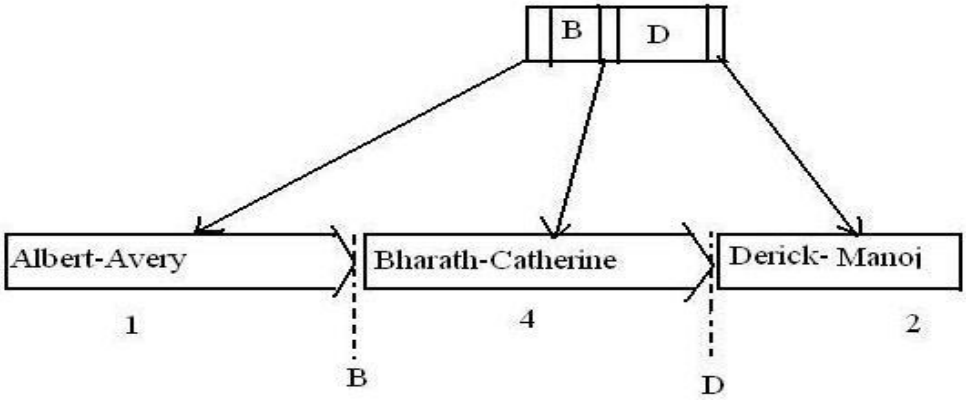
-If records are redistributed between the blocks in sequence set, value of a separator in the index set must be changed.

If split- new separator to index set.
If merged-Separator is removed.
If redistributed- Separator values are changed.

Eg1: If a record with key 'Bharath' is added to block 1 and if block 1 causes overflow, then a new separator is added. (build the index set after the sequence set is corrected)



Eg2: Suppose the record with key 'Fathima' is deleted. And block3 causes an underflow. Suppose merging tables place here, then changes are-



The index set id also changed.

Index Set Block Size:

The physical size of a node for the index set is same as the physical size of a block in the sequence set.

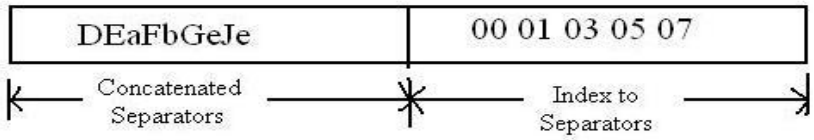
The block and index set is of same size, because-

- A common block size makes it easier to implement a buffering scheme to create a virtual simple prefix B+Tree.
- The index set block and sequence set block are often put in the same file, so that in a single access, both the data can be retrieved.
- The block size for sequence set is usually chosen because there is a good fit among this block size.

Internal Structure of Index Set Blocks: A Variable-Order B-Tree

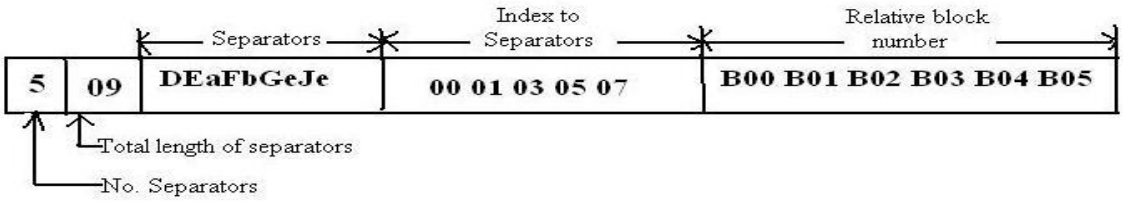
Given a large fixed size block for the index set, how are the separators stored in it? The separators are actually concatenated and put together, then there is an index to separators.

Eg: Suppose the set of separators are D, Ea, Fb, Ge &Je. These separators are concatenated and build an index for them as shown-



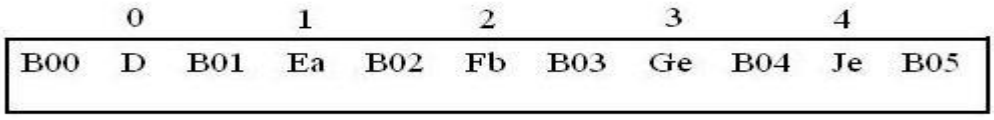
There are many ways to combine the list of separators, the index to separators and relative block numbers (RBNs) into a single index set blocks.

-One possible approach is



1) The no. of separators helps to find the middle element in the index of separators and helps in binary search.
2) Length of separators helps to move to the beginning of index to separators'.

Figure below shows the conceptual relationship of separators and the RBNs.



Suppose a record with key 'Fathima' is searched using binary search, we can conclude that it falls between the separators 'Ea' and 'Fb'. So we can come to the conclusion that record 'Fathima' is present in B02-block2 only.

This kind of index block structure illustrates 2 imp points-

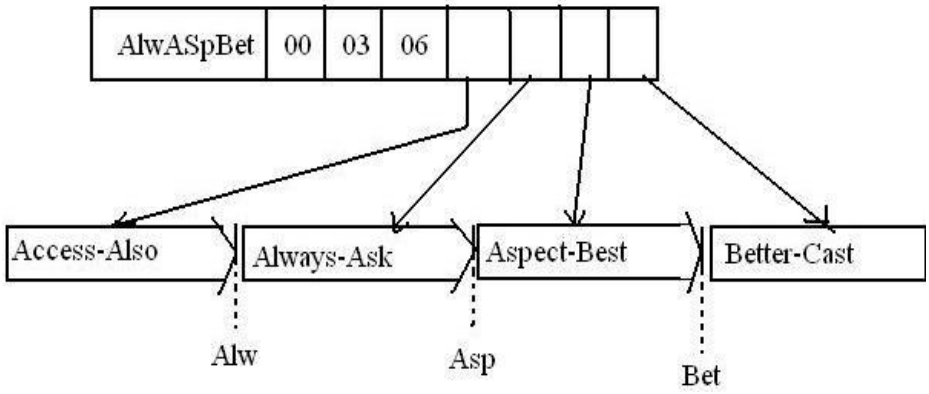
1) A block can have a sophisticated internal structure all of its own, including its own internal index, a collection of variable-length records and so forth.
2) Node of B-Tree index in simple prefix B+Tree is of variable order (since each node contains variable no. of separators of different length). This variability in size draw is to some interesting conclusions-
    a. No of separators in a block is limited by block size rather than by some predetermined order (as in BTrees of order m).
    b. Since each node is of variable order, operations like determining when a block is full or half full cannot be determined. Decisions about when to split merge, or redistribute becomes complicated.

Loading a simple prefix B+Tree-

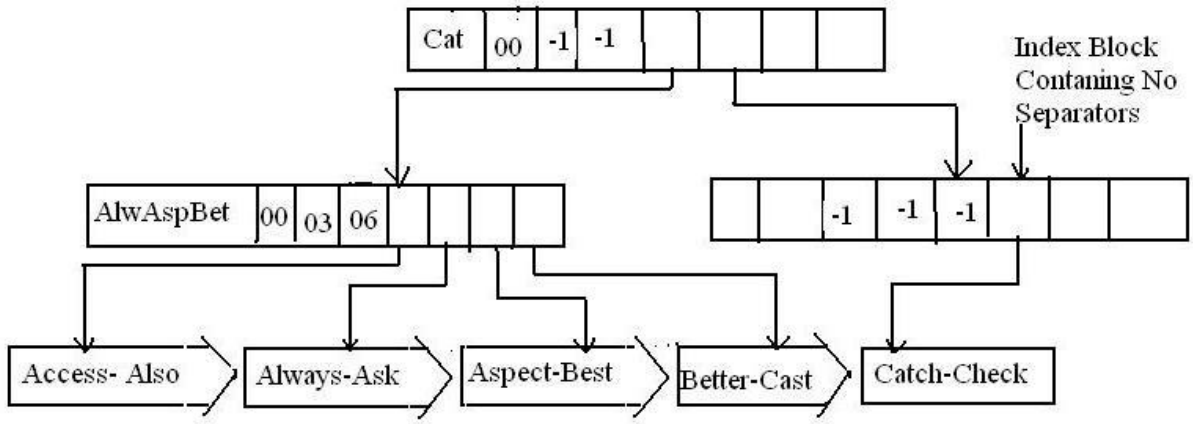
A simple prefix B+Tree is built through a series of successive or distribution of blocks. But these procedures are quite expensive. This difficulty is overcome while loading, if the records are in a sorted order. Place the records into sequence set one after the other.

A new block can be sorted when the current block fills up. Once we change the block, the shortest separator is generated and added to the block. There is no splitting or redistribution of blocks during loading of B+Tree.

Eg: The sequence set blocks and separators are loaded to memory as shown below.



Suppose the next sequence set block is Catch-Check with separator 'Cat'.



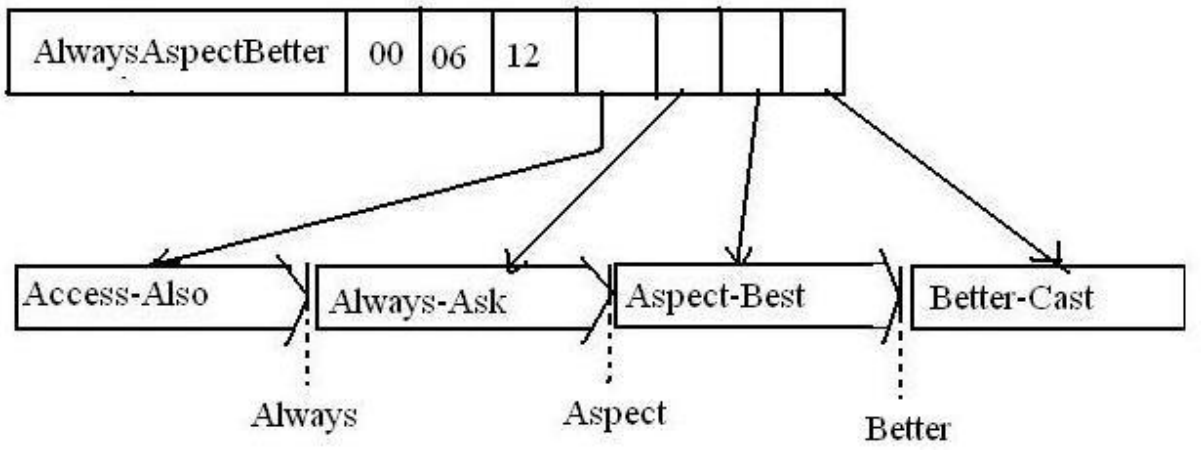
The main advantages of **simple prefix B+Tree** are-

1) The output can be written sequentially.
2) WE maybe only one pass over the data, than many passes
3) No blocks need to be reorganized as we proceed.
4) In sequential loading procedure, the trees are 100% utilized.
5) Sequential loading process creates a special locality within a file.

B+Trees:

The difference between simple prefix and plain B+Trees is that in B+Trees the prefix is not used as a separator. Instead, the separator in the index set are simple copies of the actual keys.

Eg:



Simple prefix B+Trees are often more desirable than plain b+Trees because the prefix separators takes up less space than the full keys.

Simple B+Trees are sometimes more desirable since-

- They do not need variable-length separator fields. The cost of extra overhead required to maintain and use variable-length structure is eliminated.
- Some keys cannot be much compressed, when simple prefix method is used to produce separators

Eg: 1VAIS037, 1VAIS038 and so on are difficult to compress. Prefix cannot be taken as separators, so plain B+Trees are used.

B-Trees, B+Trees, and simple prefix B+Trees in perspective:

B-Trees and B+Trees and simple Prefix B+Trees are the toosl used for file-structure design.

Common Characteristics of B, B+ and simple prefix B+Trees

1. They are all paged index structures, ie entire blocks of information are brought to memory at once. The trees tend of to be broad and shallow.
2. All 3 are height balanced trees.
3. In all 3 cases, the tree grows from the bottom up. Balance is maintained by splitting, merging an redistribution.
4. In all 3 structures, two-to-three splitting and redistribution can be used to postponed splitting.

5. All 3 approaches can be implemented as virtual tree structured in which most recently used blocks are held in memory.
6. All 3 approaches can be adapted to block structure.

Difference between various structures

B-Trees

These are multilevel indexes to data files that are entry sequenced

- Strengths: Simplicity of implementation, inherent efficiency of indexing, maximization of breadth of B-Tree.
- Weakness: Excessive seeking necessary for sequential access.

B-Trees with associated information

These are B-Trees that contain record contents at every level of B-Trees.

- Strengths: Can save space.
- Weakness: Works only when the record information is located within the B-Tree. Otherwise too many seeks are required to retrieve record information.

B+Trees

The primary difference between B-Trees and B+Trees is that in B+Trees, all the key and record information is in a linked set of blocks known as Sequence Set.

* Indexed access to this sequence set is provided through a conceptually (not physically) structure called Index Set.

Advantages of B+Trees

- Sequence Set can be processed to get sequential records, ordered by key.
- Index is built with a single key or separator per block of data records. Since there are few keys, the index is smaller and shallower.

Simple Prefix B+Trees

The separators used are smaller than the keys in the sequence set.

Questions:-

1) What is sequence set? Explain how it is maintained?
2) Explain the block splitting and merging due to insertion and deletion in sequence set, with example.
3) Explain simple prefix B+Tree. Explain the issues involved in maintenance of such tree.
4) Explain the internal structure of index set blocks.
5) Compare the strengths and weakness of B+Trees and BTrees.
6) Explain the loading of simple prefix B+Trees.
7) How are separators used instead of keys?
8) Compare B+ Tree and simple prefix B+ Trees.