CS 344: DATABASE MANAGEMENT SYSTEMS

LECTURE NOTES: 28TH SEP, 2011

STORAGE STRUCTURE AND INTRODUCTION TO INDEXING METHODS

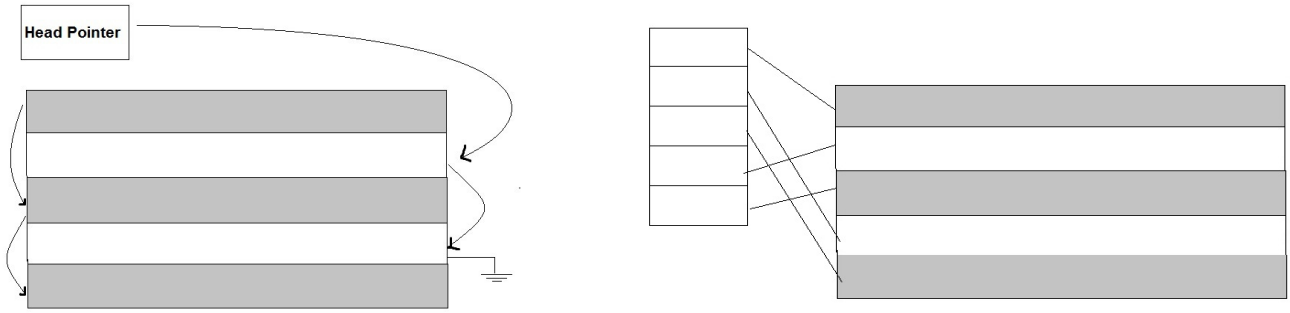
Submitted by:

Akshay Madrosiya(0910106)

Akhikesh Shah(0910147)

Comparisons of Pointers and Indexing Methods in Storage Structure and Querying:

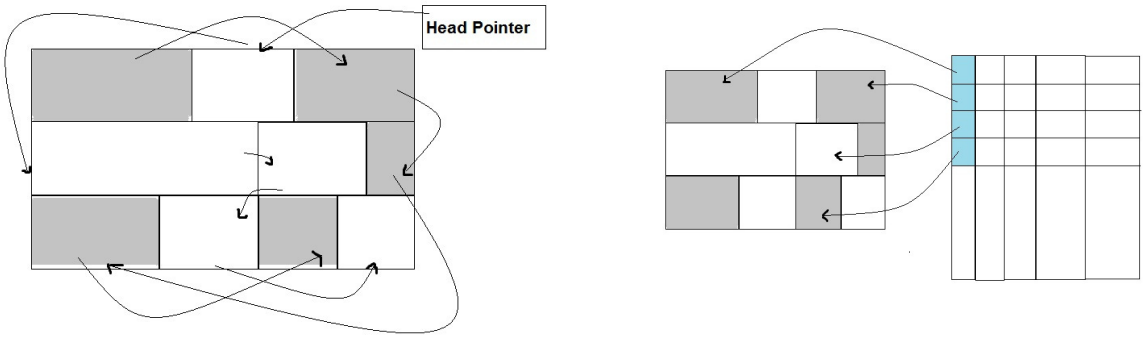
1. Fixed Length Records Files:
    - **Pointers** enable to maintain separate chains of empty and non-empty slots. Insertion becomes easier but modification and deletion still remain linear time operations since for these operations scanning the whole chain is required.
    - **Indexing** of slots is done. No performance gain is observed in modification and deletion operations since they still require searching through the index table for the particular record.



Empty Slot

Non-empty slot

2. Variable Length Record Files:
    - **Pointers** are again used to maintain chains. Hence insertion is performed in constant amount of time.
    - **Indexing**: the table consists of id, size and starting point of the slot, a flag to indicate whether the slot empty, and the pointer to that slot. Insertion involves searching for the empty slot which has sufficient size to hold the record. Modification and deletion are faster since no. Of computations are reduced in this case.



Empty Slot Non-empty slot

Hashing Techniques:

Simple/Static Hashing:

The pages containing the data can be viewed as a collection of **buckets**, with one **primary** page and possibly additional pages linked in a chain. The bucket to which the record belongs can be determined by applying a special function called hash function to the search key. A file consists of buckets 0 through $N - 1$, with one primary page per bucket initially. Buckets contain pointers to the *data entries*.

Extendible Hashing:

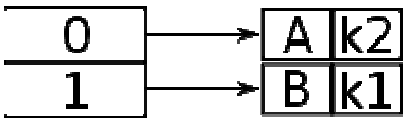
In case no. of records in any particular bucket increases, no. of bits used to determine the bucket storing the record can be increased. An addition of one bit doubles the no. of buckets i.e. splits a bucket into two subsequent buckets.

Assume that the hash function $h(k)$ returns a binary number. The first i bits of each string will be used as indices to figure out where they will go in the "directory" (hash table). Additionally, i is the smallest number such that the first i bits of all keys are different.

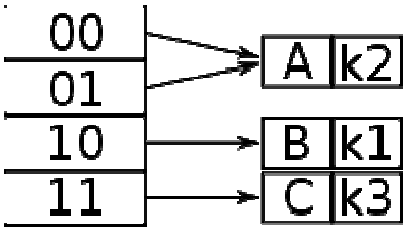
Keys to be used:

$h(k_1) = 100100$
$h(k_2) = 010110$
$h(k_3) = 110110$

Let's assume that for this particular example, the bucket size is 1. The first two keys to be inserted, k_1 and k_2, can be distinguished by the most significant bit, and would be inserted into the table as follows:



Now, if k_3 were to be hashed to the table, it wouldn't be enough to distinguish all three keys by one bit (because k_3 and k_1 have 1 as their leftmost bit. Also, because the bucket size is one, the table would overflow. Because comparing the first two most significant bits would give each key a unique location, the directory size is doubled as follows:



And so now k_1 and k_3 have a unique location, being distinguished by the first two leftmost bits. Because k_2 is in the top half of the table, both 00 and 01 point to it because there is no other key to compare to that begins with a 0.

Re-Hashing:

This is an alternate to extendible hashing. This technique uses two subsequent hash function to solve hash collisions. Insertions can trigger bucket splits, but buckets are split in sequential fashion which is the main difference when compared with extendible hashing.

*__Limitation of Hashing Techniques__- similarities between records cannot be determined.

Tree Based Indexing:

Tree-based indexing organizes the records into a single tree. Each path into the tree represents common properties of the indexed records, similar to decision trees or classification trees. The top root depicts the information in the most condensed manner and the bottom most level of the tree in the most elaborate manner.