**Logo

Description automatically generated**

**CENG307 FILE ORGANIZATION**

**HASHİNG METHODS DELETİON**

Ayben GÜLNAR-191180041

**DECEMBER 2022**

CONTENT

[FIGURES 2](#_Toc122642842)

[1.INTRODUCTION 3](#_Toc122642843)

[2. B+ TREES DELETION METHOD 3](#_Toc122642844)

[3. DYNAMIC HASHİNG DELETION METHOD 5](#_Toc122642845)

[4. LINEAR HASHING DELETION METHOD 7](#_Toc122642846)

[7.CONCLUSION 9](#_Toc122642847)

[8. REFERENCES 10](#_Toc122642848)

[9.APPENDIX 11](#_Toc122642849)

# FIGURES

[Figure 2.1 3](#_Toc122642672)

[Figure 2.2 4](#_Toc122642673)

[Figure 2.3 5](#_Toc122642674)

[Figure 3.1 6](#_Toc122642672)

[Figure 3.2 6](#_Toc122642673)

[Figure 4.1 8](#_Toc122642674)

[Figure 4.2 8](#_Toc122642672)

# 1.INTRODUCTION

B+ trees are a type of tree data structure that is commonly used to implement indexing in database systems. They are known for their ability to support efficient insertions, deletions, and searches, as well as their ability to handle large volumes of data. When deleting a record from a B+ tree, the process involves locating the record and removing it from the tree, while also ensuring that the tree remains balanced and meets the structural requirements of a B+ tree. Dynamic hashing is a technique for implementing a hash table, which is a data structure that allows for fast insertion, deletion, and lookup of data. A hash table uses a hash function to map keys to a specific location in the table, where the corresponding value can be stored or retrieved. Dynamic hashing allows the hash table to grow or shrink as needed to accommodate the number of elements being stored in it, by using a hash function that maps keys to a variable-length range of hash values. When deleting a record from a dynamic hash table, the process involves locating the record using the hash function and then removing it from the tableLinear hashing is a dynamic hashing technique that allows a hash table to grow incrementally by adding one bucket at a time as needed. When deleting a record from a linear hash table, the process involves locating the record using the hash function and then removing it from the table. In linear hashing, the deleted element may be replaced with a special "tombstone" marker to indicate that the element has been deleted, or the element may be physically removed from the table and the elements that come after it shifted to fill the gap. In summary, "B+ trees," "dynamic hashing," and "linear hashing" are all techniques for storing and organizing data in a efficient and scalable manner, and they all have specific approaches for deleting records. Understanding how these techniques handle deletions can be useful in designing and implementing data storage and indexing systems.

# 2. B+ TREES DELETION METHOD

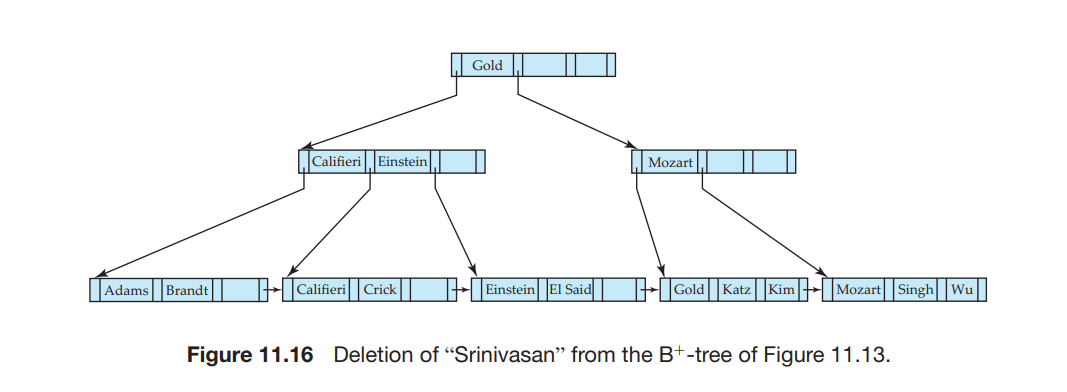


Figure 2.1

When deleting a record from a B+ tree, the resulting tree node may contain too few pointers. To fix this, the node may need to be merged with a sibling node or have its entries redistributed between the nodes. This is done to ensure that each node is at least half-full. To delete a record, we first locate the entry using a lookup algorithm and then delete it from the leaf node. If the resulting node has fewer than the required minimum number of entries, we can either merge it with a sibling node or redistribute its entries between the nodes. If we merge the nodes, we move the entries from both nodes into one of the siblings and delete the other. If we redistribute the entries, we move some of the entries from one of the siblings into the node with too few entries. After the deletion, we must also delete the entry in the parent node that pointed to the just-deleted node. In this example, the record being deleted from a B+ tree is (Srinivasan, n3), where n3 is a pointer to the leaf node containing "Srinivasan". After deleting this record, the parent node may become underfull, which means it has fewer than the required minimum number of entries. In this case, we try to merge the underfull node with a sibling node if possible. If merging is not possible, we redistribute the entries between the nodes. To redistribute the entries, we move some of the pointers from one sibling to the underfull node. However, this may cause the values separating the pointers in the underfull node to be incorrect, as they are now separated by different values than those in the parent node. In this case, we need to update the values in the underfull node and the parent node to correctly separate the pointers.

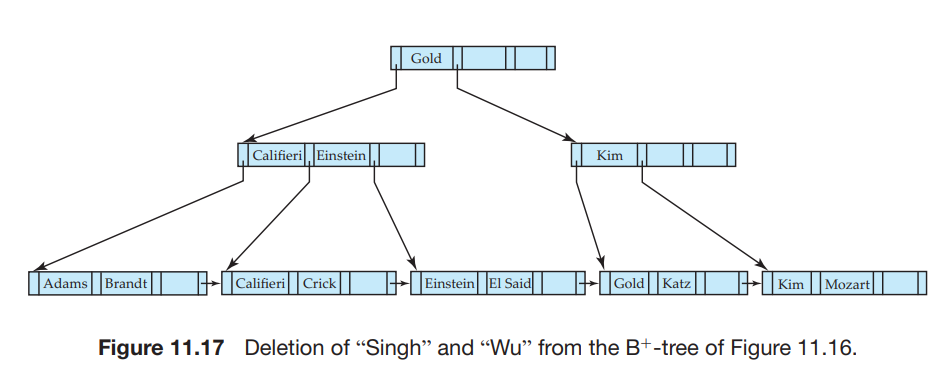


Figure 2.

In this example, we are deleting the records "Singh" and "Wu" from a B+ tree. The deletion of "Singh" does not make the leaf node underfull, but the deletion of "Wu" does. Since it is not possible to merge the underfull node with its sibling, we redistribute the values between the nodes. This involves moving the value "Kim" into the node containing "Mozart". As a result of the redistribution, the value separating the two siblings in the parent node is updated from "Mozart" to "Kim".

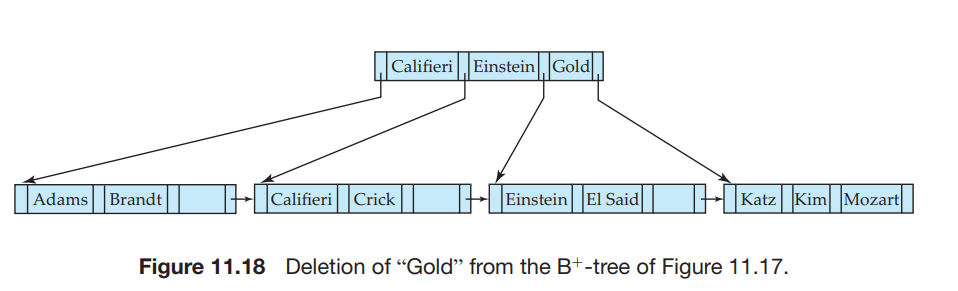


Figure 2.

In this example, we are deleting the record "Gold" from a B+ tree. This causes the leaf node to become underfull, which can be merged with its sibling. The resulting deletion of an entry from the parent node makes it underfull as well. In this case, the parent node can be merged with its sibling. This causes the value "Gold" to move down from the parent into the merged node. As a result of this merge, the root node may become underfull and need to be deleted, in which case its sole child becomes the new root and the depth of the tree decreases by 1. It is important to note that, as a result of deletion, a value that is present in a nonleaf node of the B+ tree may not be present at the leaf level. For example, in this case, the value "Gold" has been deleted from the leaf level but is still present in a nonleaf node.

To delete a value from a B+ tree, we perform a lookup for the value and delete it. If the node becomes too small as a result of the deletion, we delete it from its parent. This process may be repeated recursively until we reach the root, the parent remains adequately full after the deletion, or we need to redistribute the entries between nodes. The pseudocode provided outlines the steps for deleting a value from a B+ tree. It includes a procedure for swapping variables and a condition for determining when a node has too few pointers or values. For nonleaf nodes, this criterion is having fewer than n/2 pointers; for leaf nodes, it is having fewer than (n - 1)/2 values. The pseudocode also describes how to redistribute entries by borrowing a single entry from an adjacent node, or by repartitioning the entries equally between the two nodes. In the pseudocode, an entry is represented as (K, P), where K is the key value and P is the pointer. For leaf nodes, the pointer comes before the key value, while for nonleaf nodes, the pointer comes after the key value [1].

# 3. DYNAMIC HASHİNG DELETION METHOD

Dynamic hashing, implemented using a tree structure, allows the index to expand and contract more smoothly. Each leaf of the tree points to a bucket of data. If a bucket becomes full, the corresponding leaf becomes an internal node with two new leaves added to it, one pointing to the original bucket and the other pointing to a new bucket. If two neighboring buckets have low utilization, they can be merged into a single bucket and the corresponding leaves in the tree are removed, with the parent node now pointing to the merged bucket. The path through the tree for a particular record is determined using a hash function and a pseudorandom generator, which generates a unique binary sequence based on the record's key. This sequence is used to navigate through the tree until a leaf is reached [2].

When a bucket becomes underutilized due to deletions, it may be merged with another underutilized bucket. This involves gathering the records from both buckets into a single bucket and deleting the corresponding cells in the index list. This reduces the size of the index by removing unnecessary buckets and cells. According the othr article [3] “Deletion is handled in the obvious way by reversing the splitting process.”. *That's why I researched split and deletion in this article.*

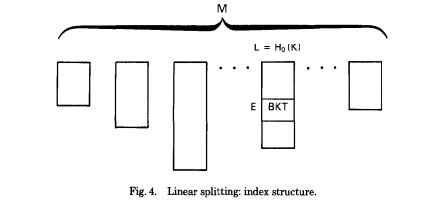


Figure 3.1

A dynamic hashing index organizes data into buckets and uses a hash function to map keys to these buckets. Initially, the index consists of M lists, each containing a single cell that points to a bucket. As records are added to a bucket, it may become full and need to be split into two smaller buckets. The addresses of these two buckets are added to the index list in new cells, and the original cell is deleted. This process repeats as more records are added and buckets become full. If the index list for a particular bucket grows to contain 2^n cells, the next bucket to be split will be the first one in the list. As the number of records in the index decreases, buckets can be merged together, with their contents being moved to a single bucket and the corresponding cells in the index list being deleted. The index also includes a supplementary cell for each list that keeps track of the number of records that map to that list. An algorithm is used to determine the specific entry in the index list for a given key [2].

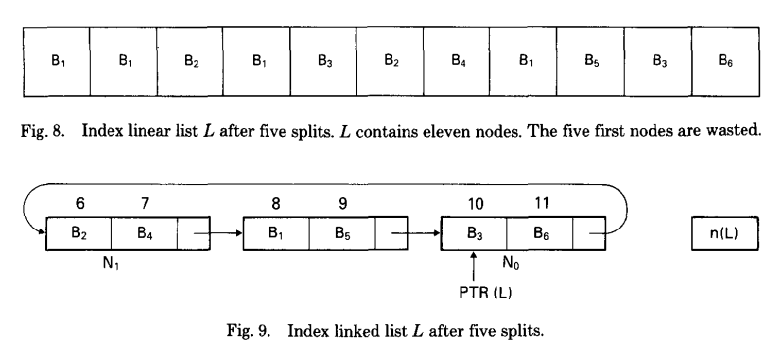


Figure 3.2

Figure 9 illustrates the implementation of list L after five splits. Each bucket is labeled by the corresponding entry in the previous linear list. To delete a record from the index, the bucket containing the record is located using the hash function and pseudorandom generator. The record is then removed from this bucket. If the bucket becomes underutilized after the deletion, it may be merged with another underutilized bucket. This involves gathering the records from both buckets into a single bucket and deleting the corresponding cells in the index list. The process of splitting and merging buckets is used to maintain a balance between the size of the index and the utilization of the buckets. The process of splitting and merging buckets is used to maintain a balance between the size of the index and the utilization of the buckets. When a bucket becomes full, it is split into two smaller buckets and the addresses of these buckets are added to the index list in new cells. When two underutilized buckets are merged, the records from both buckets are gathered into a single bucket and the corresponding cells in the index list are deleted. By dynamically splitting and merging buckets, the dynamic hashing index can adapt to changes in the size of the dataset and maintain good utilization of the buckets, which can improve the performance of the index [2]. You can review APPENDIX-1 AND APPENDIX-2.

**DELETION ALGORİTHMS:**

Dl. SEARCH; Delete (this possibly implies freeing an empty bucket);

D2. IF n(L) mod yb = 1. then MERGE

D3. n(L) +-n(L) - 1

D4. END

# 4. LINEAR HASHING DELETION METHOD

In the book [4], it is said about the deletion method, so I will talk about the insertion process first (Ramakrishnan and Gehrke, 2003):

“We not discuss deletion in detail, but it is essentially the inverse of insertion. If the last bucket in the file is empty, it can be removed and Next can be decremented. (If Next is 0 and the last bucket becomes empty, Next is made to point to bucket (AI/2) 1, where !vI is the current number of buckets, Level is decremented, and the empty bucket is removed.) If we wish, we can combine the last bucket with its split image even when it is not empty, using some criterion to trigger this merging in essentially the same way. The criterion is typically based on the occupancy of the file, and merging can be done to improve space utilization.”

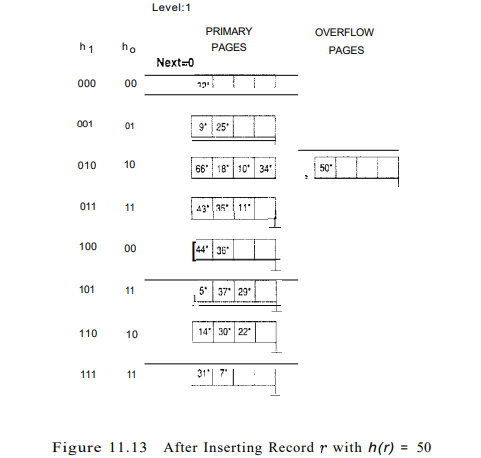
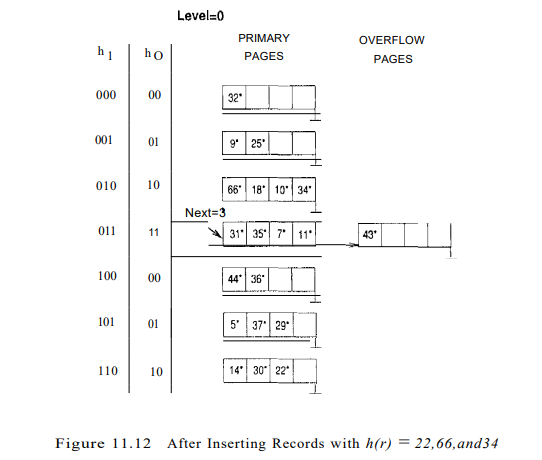


Figure 4.1 Figure 4.2

In summary for insertion, searching for a specific record in a hash table using its key (an equality selection) typically costs one disk I/O, unless the bucket containing the record has overflow pages, in which case the cost may be higher. On average, the cost of an equality selection is about 1.2 disk accesses for data distributions that are reasonably uniform. However, if the data distribution is very skewed, the cost can be considerably worse, potentially linear in the number of data entries in the file. The space utilization is also very poor with skewed data distributions. Adding a new record to the hash table (an insert) typically requires reading and writing a single page, unless a split is triggered. A split occurs when the number of records in a bucket exceeds a certain threshold, in which case the records are divided into two or more buckets to prevent the hash table from becoming too large. In this case, additional disk I/Os may be required to perform the split and update the hash function [4].

If we summarize the deletion in linear hashing; in linear hashing, deleting a record involves the following steps. Locate the record using the hash function: The hash function maps the key of the record to a specific location in the hash table. Remove the record from the table: Once the record has been located, it can be physically removed from the table. This may involve replacing the record with a special "tombstone" marker to indicate that it has been deleted, or it may involve physically removing the record and shifting the elements that come after it to fill the gap. Update the hash function if necessary: If the deletion of the record causes the load factor of the table to fall below a certain threshold, the hash function may need to be updated to adjust the size of the table. This is done to ensure that the table remains efficient and scalable as the number of records stored in it changes. Overall, the process of deleting a record in linear hashing involves locating the record using the hash function and then physically removing it from the table, while also ensuring that the hash function is updated as needed to maintain the efficiency and scalability of the table [4].

Deletion in a dynamic hashing index is similar to deletion in a static hash file, where records are removed from the file as needed. However, in a dynamic hashing index, it is possible to shrink the file when a significant number of records have been deleted in order to reduce wasted space. This is done by removing the last bucket in the data file, which contracts the file linearly. Additionally, it is possible to merge the records from a bucket with its "buddy page" (using d-1 hash bits) in order to improve space utilization. The rationale for this is that as the number of records in the file shrinks, the size of the buckets remains large, leading to wasted space. By merging buckets and contracting the file, it is possible to improve space utilization and make better use of the available storage [5,6].

# 7.CONCLUSION

In summary, linear hashing, dynamic hashing, and B+ trees are different techniques for storing and organizing data in a efficient and scalable manner. These techniques all have specific approaches for deleting records. In linear hashing, deleting a record involves locating the record using the hash function and then physically removing it from the table. The deleted element may be replaced with a special "tombstone" marker to indicate that it has been deleted, or the element may be physically removed from the table and the elements that come after it shifted to fill the gap. In dynamic hashing, deleting a record involves locating the record using the hash function and then removing it from the table. Dynamic hashing allows the hash table to grow or shrink as needed to accommodate the number of elements being stored in it, by using a hash function that maps keys to a variable-length range of hash values. In B+ trees, deleting a record involves locating the record and removing it from the tree, while also ensuring that the tree remains balanced and meets the structural requirements of a B+ tree. If the resulting tree node becomes underfull, it may need to be merged with a sibling node or have its entries redistributed between the nodes. This is done to ensure that each node is at least half-full. After the deletion, the entry in the parent node that pointed to the just-deleted node may also need to be deleted.

# 8. REFERENCES

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database System Concepts," 6th ed., McGraw-Hill Education, 2010.

[2] P. G. Selinger, "New file organization based on dynamic hashing," in ACM SIGMOD Record, vol. 13, no. 1, pp. 45-54, Mar. 1984.

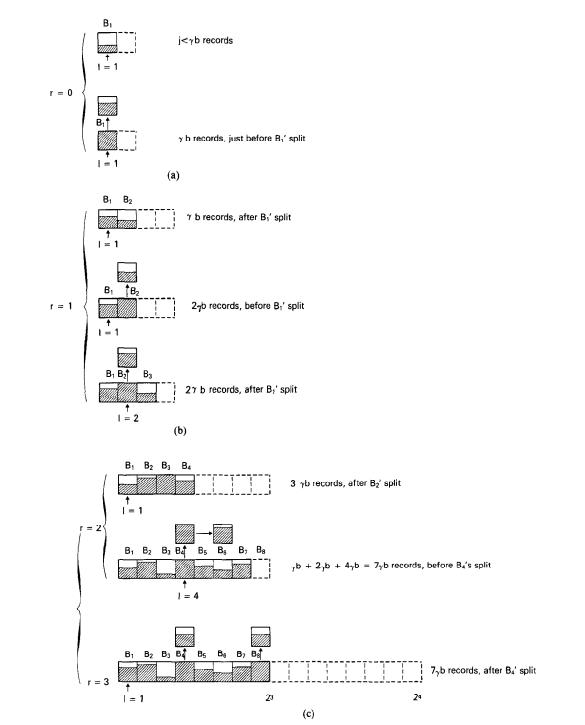
[3] K. Ramamohanarao and J. W. Lloyd, "Dynamic hashing schemes," in Department of Computer Science, University of Melbourne, Parkville, Victoria, Australia, 1984.

[4] R. Ramakrishnan and J. Gehrke, "Database Management System," 3rd ed., McGraw-Hill Education, 2003.

[5] "Linear hashing," in Department of Computer Science and Engineering, University of New South Wales, 2021, [Online]. Available:

[6] https://www.cse.unsw.edu.au/~cs9315/21T1/lectures/linear-hashing/slides.html#s13. [Accessed: 22-Dec-2022].

# 9.APPENDIX

APPENDIX-1

APPENDIX-2

