**CSE 5331: DBMS MODELS AND IMPLEMENTATION**

**FALL 2018**

**Project 1: B+ Tree Implementation**

Under the guidance of –

Prof. Sharma Chakravathy

Abhishek Santra (TA)

Submission by Team no.17

Kanthi Sannapa Komar (1001529656)

Shreyas Mohan (1001669806)

**Index**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Topic** | **Page Number** |
| 1 | Acknowledgement | 3 |
| 2 | Overall Status | 4 |
| 3 | File Description | 5 |
| 4 | Division of Labor | 6 |
| 5 | Logical Errors | 7 |

**Acknowledgement**

We would like to express special thanks of gratitude to our professor, Prof. Sharma Chakravathy who gave us the golden opportunity to do this project on B+ Tree Implementation. Through this project, our foundation on B+Tree logic became stronger and the understanding of the concept became more clearer through practical knowledge. We also like to thank our teaching assistant, Mr. Abhishek Santra who gave his valuable time in explaining how to begin with the project. Through a couple of meetings, he also cleared our doubts which helped us in completing the project successfully.

Kanthi Komar

Shreyas Mohan

**Overall Status**

In this project, we created modules that deal with the insertion and deletion of entries in the B+ Tree. We have implemented a full insert using the methods- insert () and \_insert() , and naive delete using the method ‘NaiveDelete()’ in B+ Tree Structure.

insert() method

In this method, we pass the parameters <key, recordID>. Initially, we check if the header exists or not as for the first time, B+ Tree would be empty and header would be pointing to an invalid page. We create a new leaf page, insert the entry, set the previous and next pointers. We also update the header to point to the newly created leaf page.

If the header already exists, we call the \_insert(), where the further insertion takes place and that method return a key data entry to check if the leaf split has occurred. If leaf split has occurred, the index will be updated.

\_insert() method

The \_insert() function checks whether the current page is a Leaf or an Index. In case, it is a leaf page, it checks for space in the page and inserts into it. Once the leaf page becomes full, we need to split the data into a newly created leaf. The records in the first leaf page is counted and the middle record in the first leaf page is found. We divide the counter into half and insert the second half of records into the new leaf and delete them from the first leaf page. The new key is then inserted into the page accordingly by comparing the key with last record in the first leaf page. We also need to copy up the first entry into the index above, which if not present at the very first leaf split, must be created and made to point to the two leaf pages, and also update the header page.

If it is an index page, we check if the split has occurred. If the split has occurred, we insert and update the index. In case the index is full, we split the index by copying all entries into new leaf and then rewriting the first half entries back into the old index. If the root was split, then we create a new root and move up the first entry into new root and point the header page to new root.

NaiveDelete() method

We search the leaf page containing the key to be deleted. Using an iterator, we traverse to the end of the leaf. We compare the key with the iterator key and then perform deletion. We use recursion to remove all occurrences of the key, which means the duplicate values are also deleted.

**File Description**

No additional files have been created in this project.

**Division of Labor**

We (Kanthi Komar and Shreyas Mohan) worked in this project as a team. Initially, it was an arduous task to understand the logic and approach to the implementation. It took us couple of meetings to implement the insert() function itself and insertion of entry into the leaf. Once we completed the leaf split, it was easier for us to implement the index split as well. We discussed together on understanding problem statement and working on algorithm and implementation was done by both of us. Testing was done by Kanthi Komar. Reporting and Documentation was done by Shreyas Mohan.

**Encountering Logical Errors**

1. We wrongly assumed that the right most leaf only splits and wrote the leaf split code in a wrong way. We then realized and worked on paper how exactly the leaf splits and we were able to approach to the solution.
2. We encountered a problem while splitting the leaf equally. We were calculating the middle element as (total+1)/2,where total=currentLeaf.numberOfRecords(). This was resulting in an imbalanced leaf. We solved this by assigning middle=(total)/2 and storing the record before the middle element in a variable named prevTemp, which will be compared with the key element to be inserted.
3. On deletion, we missed on writing the code for deleting multiple values. Then we used recursive function to remove multiple occurrences of the key to be deleted.
4. We faced ArrayIndexOutOfBoundsException on deleting some records like the last record in the leaf of B+ tree. The error occurred at the line where we were checking for multiple occurrences, as the next pointer of the last leaf was pointing to invalid page. We solved this by checking the condition if the next page ID is a positive value.