Database Models and Implementation

Project Report

Student Details:

Name: Vishnu Gopal Rajan

Id: 1001755911 **Project Group : 12**

➢ Overall Status:

- Major components of this project milestone included coding of *Insert(), _insert() and Naivedelete()* functions. These functions included creating of respective nodes depending on the requirement and inserting record or key into them respectively and also deleting the record entry by searching for it in the B+ tree.
- **Insert() and _insert():**
  - The first challenge was to check if the tree was empty or not using headerPage.get_rootId. If tree was empty, a new BTLeafPage object to create a new leafpage had to be created and add record into it. Also the pointers to this node had to be pointing to INVALID_PAGE.
  - If there existed a tree already then we had to call the _insert() function which would check the respective page, if it was an Index or Leaf page and insert the record accordingly.
  - To check the page type, a BTSortedPage object had to be created. If the page was a Leaf page then, the available space on the page had to be checked using *curLeafPage.available_space() >= BT.getKeyDataLength(key, NodeType.LEAF)* . If there was enough space for the record then it would be inserted in the leaf page. Else we would have to split the leaf page into 2 and divide the records in it equally by:
    - Moving all records to newly created leaf page
    - Moving the records from the new leaf page to current leaf page until both are equally filled (half) using *newLeaf.available_space() < curLeafPage.available_space()*.
    - Copy the first record value in the new leaf page to the index page.

        - Compare key value with first value in new leaf and insert accordingly
    - For index page as well we follow the same routine recursively, where the available space in the index node is checked using *curIndex.available_space() >= BT.getKeyDataLength(copyUp.key, NodeType.INDEX.* If there is enough space to insert key then it will inserted in the current index page else index gets split equally and the first record in the new index will be pushed up to the new root (not copied).
        - Moving all records to newly created index page
        - Moving the records from the new index page to current index page until both are equally filled (half) using *curIndex.available_space() > newIndex.available_space()*
        - Copy the first record value in the new index page to the root page.
        - Compare key value with first value of new index page and insert accordingly.
- **Naivedelete():** This function enables us to delete either just a record or a range of records.
    - Basically at the start we use *leafpage = findRunStart(key, tmprid);* to get the left most leaf inorder to check for the record in all the leaf nodes connected via doubly linked list. We first check if page exists, if it does then we carry on to getCurrent records.
    - Then we check if record exists, if not then we check the adjacent pages using getNextPage() , if no next page also exists then we return false (No record found to delete).
    - If the record exists then we move on to finding it on the page using (*BT.keyCompare(key, record.key)<=0*). Once its found we delete the record using *leafpage.delEntry(k).* If the range of records range for more than one page then we need to check for these records in the adjacent page as well. Thus using getNextPage() to get the next page and check it as well.

➢ FILE DESCRIPTIONS:

- KeyDataEntry newRootEntry : Used to enter new root entry into the newly created root node

- BTIndexPage newRootPage; and PageId newRootPageId: Used to create new index page called new Root page along with its pageId called newRootPageId to accommodate index keys during leaf split.
- BTSortedPage curPage : This is an object of BTSortedPage used to check if page is Index page or Leaf page.
- BTLeafPage newLeaf; and PageId newLeafId: Used to create new leaf page called newLeaf and pageId NewLeafId to accommodate records during leaf split.
- BTIndexPage newIndex; and PageId newIndexId: Used to create new index page called newindex and pageId newIndexId to accommodate keys during index split.
- PageId childPtr: Used to hold the pageId of the child pointer.
- KeyDataEntry copyUp: Used to store the values or entry to be copied up or moved up to the next level of the B+tree.
- KeyDataEntry record: Used to hold the record entry to be deleted.

➢ DIVISION OF LABOR:

I am individually doing the project. I have spent approx. 15 hours+ on this project. Most of the time was spent of trying to getting the right file descriptions and correct errors.

➢ LOGICAL ERRORS:

- newLeaf.available_space() < curLeafPage.available_space(): This process of splitting the nodes equally was actually quite simple but I tried much more complex logics which included finding the number of bytes etc. which did not give the correct outputs. I also tried to find the space occupied by each record in order to compute the total space of the leaf node, which made it more complicated. Thus I finally solved it using just the available_space() method.
- Another logical error I faced was trying to figure out the way to get the child pointer pageId. I tried multiple methods which were incorrect and gave errors. Since I was using eclipse IDE I could get a prompt to help me with casting. Finally I used the getData() method to derive the pageId of the record to use as childPointer.

- Another logical error I faced during coding the delete method included not traversing to the next page when doing a range delete. This got me stuck only to the first page. After going through the findRunStart() function I could understand how I had to jump to next page using recursive while loop and perform the delete operation recursively.
- I also faced many other logical errors like mismatch of objects, rid and key values.