Lab 2 **Hanjing Wang(516030910015)**
Database System ACM Class, Zhiyuan College, SJTU
Prof. **Feifei Li** Due Date: May 5, 2019

Design Decisions

simpledb.BufferPool

Page Eviction Algorithm

In the practice of Lab 2, I choose Random Eviction among all the possible workarounds for its simplicity, which means that the implementation of Random Eviction does not have additional cost to maintain other data structures like queue for FIFO(First In First Out) algorithm. Therefore, the algorithm is more efficient in the settings that not all the visits are limited to a small page set and the cache size is relatively small. The settings actually make sense in the real application.

Trim Cache

This method merges local cache into the global one. Otherwise the buffer is risky since the corresponding pages in the global cache may have been evicted, therefore having potential loss of updates.

Insertion/Deletion Method

The **BufferPool.insertTuple/deleteTuple** method calls the **BTreeFile.insertTuple/deleteTuple** method, and then synchronizes the local cache with global cache in the **BufferPool**.

simpledb.BTreeFile

splitLeafPage

First, the method get an empty page from the **getEmptyPage** method and formats it as a new leaf page at the right of the existing one. Then a reverse iterator return the right most tuple in the leaf page iteratively, delete it at the original page and insert it at the new page. This delete-and-insert method is not optimal for performance since we could do a deep copy of part of the page and paste it to the new one. However, in that way we still have to visit every tuple to update the occupation of each slot. And there is potential performance advantage for insertion operation later(time locality) because the tuples are well orgnized. Parent pages are recursively adjusted(splitting or moving entries) to accommodate the new entry. At the end, the sibling and parent pointers are updated.

splitInternalPage

Similar to the **splitLeafPage** method. The major difference lies in parent entry key. Instead of copying the key of the left most tuple in the new page like **splitLeafPage**, the middle key is deleted from the original page and inserted into the parent page.

stealFromLeafPage

This method is quite similar to the **splitLeafPage** method in implementation, for both of them delete the left/right most tuples and insert them in the another page iteratively and finally update the parent entry that refers to the pages.

stealFromInternalPage

Both of the stealing from internal page method actually "rotate" the entries by first pull down the key from the parent entry and insert it as a new entry in the target page with delivery of child pointer from another entry originally locating in the source page next to it. The key of the entry giving out its child pointer is pushed to the parent entry. Therefore, in the rotation one key is pushed up and one key is pulled down. Parent pointer of every moved tuple is updated at last.

Difficulties

I spent 3 days in this lab work, a little longer than the previous one. I ingored the synchronization between local cache and global cache at the very begining because pages opened with "Read-Write" permissions were recorded in the global cache. Therefore I thought it was unnecessary to synchronizes local cache and global cache explicitly. However, chances are that pages in the local cache have already been evicted during the operation procedure. Then a dirty page in local cache may never have the oppotunity to write the modification on it back into disk for the **flushback** method only applies to pages in global cache. I had a hard time locating the bug especially when I used a random page eviction algorithm.