Project 3 Report

1. Overview

In this project, we have implemented B+ Tree as **index layer** service for our database engine. Index layer, organized by specific attribute as the key, is used to give facilities of quick *insert/ delete/scan* of record RID entries.

2. Design & Implementation

- **B+ Tree Node format**

In the B+ Tree, there are two different kinds of nodes: non-leaf node and leaf node. We use two different class definition for them, `BTreeNode` and `LeafNode` respectively. Their on disk format is like this:

BTreeNode:

child[0]	key[0]	child[1]	key[0]	child[1]
key[0]	...	...	child[N]	key[N]
child[N+1]				
depth	type	rightid	leftid	N

LeaNode:

key[0]	rid[0]	key[1]	rid[1]	key[2]
rid[2]	...	...	key[N]	rid[N]
type	overflowID	rightid	leftid	N

To make synchronization between node update with disk content, I come up such plan: class constructor receives `FileHandle` and `pageID,` loading all information into corresponding field. Moreover, during deconstructor, the instance of node class automatically dump all its data into disk. By doing this, I don't have to bother when to update to disk.

- ***Overflow Leaf Node Page***

To handle same key entries spanning over multiple pages, I have special design for leaf node: gives it a overflow page ID. The idea is like this: each time we check whether it needs to split, if its all keys are the same, we try to spill one ***(key,rid)*** pair into its overflow page. If overflow page is also full, it would spill to next level overflow page, recursively.

So the overall B+ Tree looks like this:

However, after change to this design, it requires corresponding change for scan: during scan, instead of just looking at certain leaf node its own entries, we should retrieve all key entries from overflow pages chain.

- ***Helper Function***

For code reusing, we define some template function, operating on vector<class T>.

```
template <typename T>
vector<T> concate_vectors(vector<T> x, vector<T> y) {
   vector<T> z(x);
   z.insert(z.end(), y.begin(), y.end());
   return z;
}
```

3. Tests & Debugging

The implementation pass all tests, including extra credit cases, after some debugging. Programming is fun, debugging is not. : (

In last project report, I mentioned that, gdb tool chain is useful for debug. But now I think the most powerful debugging tool is your mind: the symptom of the problem sometimes is quite far from its root source, which requires our own analytical efforts to identify most possible reason of bugs.

Taking following bug as example: During Leaf Node dump(), some times it is called before it is fully splitted, which means its size may be larger than PAGE_SIZE, 4KB. Then buffer overflow writing causing modify other LeafNode member field, like FileHandle _fh. The consequence of

that is it’s unable to use ***PagedFileManager*** ‘s facilities.