Database Programming Project
Project 2: B+-tree
Due: June 14. 23:59 pm

1. Introduction

In this programming project, you will implement a B+-tree in which the node structure is a slotted page structure that you implemented in the last project. We also use the form of the record as variable string key (type char*) and 8bytes integer value. You will implement three functions, split(char *key, uint64_t value, char** parent_key) function in page.cpp file and insert(char* key, uint64_t), find(char *key) functions in btree.cpp.

이번 프로그래밍 프로젝트에서는 B+-tree를 구현합니다. B+-tree는 이전 프로젝트에서 구현한 하나의 Slotted Page를 하나의 B+-tree Node로 사용합니다. 이번 프로젝트에서도 마찬가지로 Record는 Variable Length Key를 사용하며, 자료형은 char*입니다. 그리고 Value로는 8byte integer (uint64_t) 자료형을 사용합니다. 이번 프로젝트에서는 총 3개의 함수를 구현하게 됩니다. page.cpp 파일에 있는 split(char* key, uint64_t value, char **parent_key) 함수와, btree.cpp 파일에 있는 insert(char *key, uint64_t), find(char *key) 함수를 구현하게 됩니다.

2. Getting Started.

Please use the source code that you used in project 1. A simple Makefile is provided for you to compile the project. To compile your project, type “make btree”. If you want to delete object files, you can delete them by typing “make clean”.

- slot_header.hpp/slot_header.cpp: In the codes, you can find the definition and implementations of the class for the slot_header structure. Please do not modify these codes.

- page.hpp/page.cpp: You will implement split function in page.cpp. Please do not modify page.hpp file.

- btree.hpp/btree.cpp: You will implement insert and find fuction in btree.cpp file. please do not modify btree.hpp file.

- main_btree_page.cpp: This file includes the main function that you can test your implementation.

If you want to test, you can execute ./btree file after compiling the source codes.

소스 코드는 project 1 진행시 사용하신 코드 그대로 사용하시면 됩니다.
Makefile: 컴파일 하실 때 사용하실 파일입니다. 이번 프로젝트를 위해서는 "make btree"라고 명령어를 치시면 됩니다. 만약 중간에 생성되는 object 파일을 지우시고 싶으실 경우, "make clean" 명령어를 치시면 됩니다.

- slot_header.hpp/slot_header.cpp: 이 파일에서는 slotted_header와 관련된 구현과 자료구조 정의등을 확인하실 수 있습니다. 이 파일을 고치지 마세요.

- page.hpp/page.cpp: page.cpp내의 insert, find, is_full함수를 이번 프로젝트에서 구현하시게 됩니다. page.hpp파일을 수정하지 마세요.

- btree.hpp/btree.cpp: btree.cpp 파일 내의 insert, find 함수를 구현하시게 됩니다. btree.hpp 파일을 수정하지 마세요.

- main_btree.cpp: 이 파일에는 main 함수가 포함되어 있습니다. 따라서, 이 파일을 사용해서 이번 프로젝트에서 작성한 코드를 테스트할 수 있습니다.

구현한 부분을 테스트 해보고 싶은 경우, 컴파일 후 생성되는 ./btree 파일을 실행하시면 됩니다.

3. Design Overview

The B+-tree is the most widely used index structure in database systems. In this project, B+-tree employs a slotted page structure as a node structure, so every inserted record is stored in the slotted page structure. When a node is complete, the node should be split into two nodes. Also, if the overflown node is the root node, then you should increase the height of the tree by making a new root. There is a variable called leftmost_ptr in the page structure in the page.hpp. The leftmost_ptr will be used for the internal node. A sibling_ptr is also in the slotted header class, but in this project, we will not consider it since there is no range query function in our project.

B+-tree는 데이터베이스 시스템에서 가장 널리 사용되는 인덱스 자료구조입니다. 이번 프로젝트에서는 project 1에서 구현한 slotted page structure를 노드 구조로 갖는 B+-tree를 구현하고자 합니다. 모든 데이터는 slotted page 자료구조에 저장됩니다. 만약 노드가 가득차게 될 경우, split연산을 통해 2개의 노드에 엔트리가 나누어져서 저장되어야 합니다. 만약 root 노드가 가득차서 split이 발생한 경우, 새로운 root 노드를 만들어주게 되며, 이에 따라 B+-tree의 높이/레벨은 증가하게 됩니다. page 자료구조에는 leftmost ptr이라는 변수가 존재하는데, internal node의 차일드 노드 중 가장 작은 키들을 가지고 있는 차일드 노드의 주소를 저장하는 데 사용됩니다. sibling_ptr 또한 slotted header에 저장되어 있지만, 이번 프로젝트에서는 range query에 대해 구현하지 않기 때문에 고려하지 않습니다.

The following functions are provided (Do not modify them!).

In page.cpp

- uint16_t page::get_type()
 This function returns the type of the page (B+-tree node)
 It would be LEAF (1) or INTERNAL (2)

 - void page::set_page_type(uint16_t)
 This function is used for setting the type of the page (B+-tree node)
 The given parameter should be LEAF (1) or INTERNAL.

 - page *page::get_leftmost_ptr()
 This function is used for getting the address of the child node, which has the smallest keys.

 - void page::set_leftmost_ptr(page *p)
 This function is used for setting the value of the leftmost ptr. It is typically used in the case of root node split.

 - void page::defrag()
 This function is used for defragmentation of the page when the node split happens.

다음의 함수는 제공되는 함수로써 수정하지 마시길 바랍니다.

- uint16_t page::get_type()
 이 함수는 page의 타입을 구할 때 사용합니다. 값은 page.hpp에 정의된 LEAF (1) 이나 INTERNAL (2) 중 하나 입니다.

- set_page_type(uint16_t)
 이 함수는 page의 타입을 설정할 때 씁니다. 이 때 값은 LEAF (1) 또는 INTERNAL (2) 중 하나여야 합니다.

- page *page::get_leftmost_ptr()
 이 함수는 페이지의 leftmost_ptr을 구할 때 쓰는 함수입니다.

- void page::set_leftmost_ptr(page *p)
 이 함수는 페이지의 leftmost_ptr을 설정할 때 쓰는 함수입니다.

- void page::defrag()
 이 함수는 split 함수 사용 후에 복사된 record들을 기존에 page에서 삭제하는 데 사용합니다.

The following functions that you will be responsible for implementing.

In page.cpp
- uint64_t page::find(char *key)
Modify the find function in page.cpp to support the traverse operation for the internal node. For example, when the reader finds the larger key, the reader should move to the child node.

- page* page::split(char *key, uint64_t val, char** parent_key)
In the split function, you should find the parent key of the newly created node and store it in the address space of parent_key. This function will return the address of the newly created node.

- void page::defrag()
Please fix the error in the defrag() function.
// code

```
new_page->set_leftmost_ptr(get_leftmost_ptr());

memcpy(this, new_page, sizeof(page));
hdr.set_offset_array((void*)((uint64_t)this+sizeof(slot_header))); // ADD this
delete new_page;
```

In btree.cpp
void btree::insert(char *key, uint64_t val)
This function traverses the btree to find the targe leaf node using the given key. If there is enough space in the leaf node, then the key and value pair are stored on the node. If there is no enough space, the split operation is triggered. The split operation can be propagated until it finds the node which is not full.

uint64_t btree::lookup(char *key)
This function traverses the btree to find the given key and returns the value. If it couldn't find the key from the btree, it will return 0.

아래의 함수는 이번 프로젝트에서 구현해야하는 함수들입니다.

In page.cpp

-uint64_t page::find(char *key)
B+-tree traverse를 위해서, page.cpp 파일에 find 함수를 internal node에 대해서도 동작할 수 있도록 고쳐야 합니다. 예를 들면, 진행하다가 찾고자 하는 키보다 큰 키를 만나게 되는 경우, 더 진행하지 않고 child node로 가야 합니다.

- page* page::split(char *key, uint64_t val, char** parent_key)
Split 함수에서는 새로운 노드를 만들고, 새로운 노드에 절반의 엔트리를 복사해야 합니다. 그 후, defrag() 함수를 불러서 복사된 엔트리를 제거해야 합니다.
split 연산을 하면서 medium 값을 구하고, medium 값의 주소를 parent_key 변수에 저장해야 합니다. 그리고 새로 생긴 노드의 주소를 리턴합니다.

- void page::defrag()
page.cpp 파일 내의 void page::defrag() 함수 내에 아래의 라인을 추가하기 바랍니다.

```
new_page->set_leftmost_ptr(get_leftmost_ptr());

memcpy(this, new_page, sizeof(page));
hdr.set_offset_array((void*)((uint64_t)this+sizeof(slot_header))); // ADD this
delete new_page;
```

In btree.cpp

- void btree::insert(char *key, uint64_t val)
이 함수에서는 먼저 target leaf 노드를 찾기 위해 root 노드부터 leaf node까지 traverse를 합니다. 만약 target leaf node에 충분한 공간이 있다면, 저장한 후 종료하고, 만약 없는 경우, split 연산을 실행합니다. 이 때, 상위 레벨의 노드에 충분한 공간이 없는 경우 split 연산이 연속적으로 발생하게 됩니다.

- uint64_t btree::lookup(char *key)
이 함수는 주어진 키를 찾기 위해 btree의 traverse를 진행합니다. 키를 찾은 경우 연관된 value 값을 리턴하고, 못 찾은 경우 0을 리턴하게 됩니다.

4. Submission Guidelines

Please submit your page.cpp and btree.cpp file to the e-Campus.
If you have any questions, please upload your question to the e-campus.

page.cpp파일과 btree.cpp 파일을 e-Campus를 통해 제출해주시고, 질문 사항 또한 e-Campus를 통해 올려주시기 바랍니다.