EduBtM Project 보고서

20180038 박형규

1. 서론

이번 project에서는 수업시간에 배운 B+ tree를 구현해보도록 하겠다, B+ tree는 데이터베이스에서 데이터를 저장하는 유용한 자료 구조로 Leaf node와 Internode(Root node 포함)으로 구성되어 있다.

2. 함수분석

2.1 API Functions

- EduBtM_CreateIndex()

이 함수는 Index file에서 새로운 B+ tree index를 생성하고, 생성된 index의 root page의 page ID를 반환하는 함수이다. 먼저 page를 할당 받고, edubtm_InitLeaf()로 할당 받은 page를 root page로 초기화하였다. 그 후 초기화된 root page의 page ID를 반환받았다.

-EduBtM_DropIndex()

이 함수는 index file에서 B+ tree index를 삭제하는 함수이다. edubtm_FreePages() 함수를 통해 B+ tree index의 root page 및 모든 자식 page들을 각각 deallocate하였다.

-EduBtM_InsertObject()

이 함수는 B+ tree index에 새로운 object를 삽입하는 함수이다. edubtm_Insert() 함수를 통해 새로운 object에 대한 key, ID pair을 Index에 삽입하였다. 그 후 root page에서 split이 발생하여 새로운 root page 생성이 필요하다면, 즉 lh == TRUE라면 edubtm_root_insert()를 호출하여 이를 처리

하였다.

-EduBtM_DeleteObject()

이 함수는 B+ tree index에서 object를 삭제하는 함수이다. 먼저 edubtm_Delete 함수를 통해 key, ID pair을 삭제한다. 그 후, root page에서 underflow가 발생하였다면 btm_root_delete() 함수를 통해 처리하였다. 다음으로 root page에서 split이 발생한 경우, edubtm_root_insert()를 호출하여 처리하였다.

-EduBtM_Fetch

이 함수는 B+ tree index에서 검색 조건을 만족하는 첫 번째 object를 검색하고 이의 cursor를 반환한다. 먼저 startCompOp가 SM_BOF라면 edubtm_FirstObject를 통해 가장 작은 key값을 갖는 leaf index를 검색한다. 다음으로 startCompOp가 SM_EOF라면 가장 큰 key값을 갖는 leaf index를 검색한다. 이외의 경우에는 edubtm_Fetch()를 호출하여 검색 조건을 만족하는 첫 번째 leaf index를 반환한다.

-EduBtM_FetchNext

이 함수는 검색 조건을 만족하는 현재 object의 다음 object를 검색하고, 검색된 object를 가리키는 cursor를 반환한다. 먼저 edubtm_FetchNext()를 호출하여 검색 조건을 만족하는 다음 leaf index를 검색하였다.

2.2 Internal Functions

-edubtm_InitLeaf()

이 함수는 page를 leaf page로 초기화하는 함수이다. Page의 header에서 type, pid, nSlots, free, unused, prevPage, nextPage등을 초기화한다.

-edubtm_InitInternal()

이 함수는 page를 internal page로 초기화하는 함수이다. Page의 header에서 pid, flags, type, p0, nSlots, free, unused 등을 초기화한다.

-edubt,_FreePages()

이 함수는 index page를 deallocate하는 함수이다. 이는 파라미터로 주어진 page의 모든 자식 page들에 대해 재귀적으로 edubtm_FreePages()를 호출하여 해당 page들을 deallocate한다.

-edubtm_Insert()

이 함수는 parameter로 주어진 page를 root page로 하는 index에 새로운 object에 대한 Key, ID pair을 삽입하고, root page에서 split이 발생한 경우, split으로 생성된 새로운 page를 가리키는 internal index entry를 반환한다. 먼저 주어진 page가 internal page라면 새로운 object를 삽입할 leaf page를 찾기 위해서 다음으로 방문할 child page를 결정한다. 그리고 이 자식 page에 새로운 object를 삽입하기 위해 재귀적으로 edubtm_Insert()를 호출하였다. 만약 결정된 자식 page에서 split이 발생한 경우, 해당 split으로 생성된 새로운 page를 가리키는 internal index entry를 파라미터로 주어진 root page에 삽입한다. 만약 파라미터로 주어진 root page에서 split이 발생하였다면 해당 split으로 생성된 새로운 page를 가리키는 internal index entry를 반환한다. 다음으로 파라미터로 주어진 root page가 leaf page라면 edubtm_InsertLeaf()를 호출하여서 해당 page에 새로운 object를 삽입하였다. 만약 split이 발생하였다면 해당 split으로 생성된 새로운 page를 가리키는 internal index entry를 반환하였다.

-edubtm_InsertLeaf()

이 함수는 leaf page에 새로운 index entry를 삽입하고, split이 발생한 경우, split으로 새엉된 새로운 leaf page를 가리키는 internal index entry를 반환한다. 먼저 새로운 index entry의 삽입 위치를

결정한다. 그리고 새로운 index entry 삽입을 위해 필요한 free area의 크기를 계산하였다. 다음으로 page에 여유 영역이 있는 경우, 필요 할 때는 page를 compact 하고, 결정된 slot 번호로 새로운 index entry를 삽입하였다. 다음으로 page에 여유 영역이 없다면 이는 page overflow가 발생한 경우이므로 edubtm_SplitLeaf()를 호출하여 page를 split하였다.

-edubtm_InsertInternal()

이 함수는 Internal page에 새로운 index entry를 삽입하고, split이 발생한 경우 split으로 생성된 새로운 Internal page를 가리키는 internal index entry를 반환한다. 이를 위해 먼저 새로운 index entry 삽입을 위하여 필요한 free area의 크기를 계산하였다. 다음으로 page에 여유 영역이 있는 경우 필요한 경우 page를 compact하고, 파라미터로 주어진 slot 번호의 다음 slot 번호로 새로운 index entry를 삽입하였다. 다음으로 page에 여유 영역이 없는 경우에는 page overflow가 발생한 경우이므로 edubtm_SplitInternal()을 호출하여 page를 split하였다.

-edubtm_SplitLeaf()

이 함수는 overflow가 발생한 leaf page를 split하여 파라미터로 주어진 index entry를 삽입하고, split으로 생성된 새로운 leaf page를 가리키는 internal index entry를 반환한다. 이를 위해 먼저 새로운 page를 할당 받고 그 page를 leaf page로 초기화하였다. 그 후, 기존 index entry들 및 삽입할 index entry를 key 순으로 정렬하여 overflow가 발생한 page 및 할당 받은 page에 나누어 저장하였다. 다음으로 할당 받은 page를 leaf page들간의 doubly linked list에 추가하고 할당 받은 page를 가리키는 internal index entry를 생성하였다. 마지막으로 split된 page가 ROOT일 경우, type을 LEAF로 변경하고 생성된 index entry를 반환하였다.

-edubtm_SplitInternal()

이 함수는 overflow가 발생한 internal page를 split하여 파라미터로 주어진 index entry를 삽입하고 split으로 생성된 새로운 internal page를 가리키는 internal index entry를 반환하는 함수이다 .

이를 위해 먼저 새로운 page를 할당 받고 그 page를 internal page로 초기화하였다. 그리고 기존 index entry들 및 삽입할 index entry를 key 순으로 정렬하여 overflow가 발생한 page 및 할당 받은 page에 나누어 저장하였다. 다음으로 split된 page가 ROOT일 경우, type을 INTERNAL로 변경하였다.

-edubtm_root_internal

이 함수는 Root page가 split된 색인을 위한 새로운 root page를 생성하는 함수이다 . 먼저 새로운 page를 할당 받고, 기존 root page를 할당 받은 page로 복사한다. 그리고 기존 root page를 새로운 root page로서 초기화하였고 할당 받은 page와 root page split으로 생성된 page가 새로운 root page의 자식 page들이 되도록 설정하였다.

-edubtm_Delete()

이 함수는 파라미터로 주어진 page를 root page로 하는 색인에서 object를 삭제한다. 먼저 파라미터로 주어진 page가 internal page라면 삭제할 object를 찾기 위해 다음으로 방문할 자식 page를 결정한다. 그리고 결정된 자식 page를 root page로 하는 subtree에서 재귀적으로 edubtm_Delete()를 호출하였다. 마지막으로 결정된 자식 page에서 underflow가 발생한 경우, btm_Underflow()를 호출하여 이를 처리하였다. 다음으로 파라미터로 주어진 root page가 leaf page인 경우 edubtm_DeleteLeaf()를 호출하여 해당 page에서 object를 삭제하였다.

-edubtm_DeleteLeaf()

이 함수는 leaf page에서 object를 삭제하는 함수이다. 먼저 삭제할 object가 저장된 index의 offset이 저장된 slot을 삭제하였다. 그 후 leaf page의 header을 갱신하고 underflow가 발생한 경우 f를 TRUE로 설정하였다.

-edubtm_CompactLeafPage()

이 함수는 Leaf page의 데이터 영역의 모든 free area가 연속된 하나의 continuous free area가 되도록 index entry들의 offset을 조정하는 함수이다. 먼저 파라미터로 주어진 slotNo가 NIL이 아니라면 slotNo에 대응하는 page의 모든 index entry들을 데이터 영역의 가장 앞부분부터 연속되게 저장하였다. 다음으로 파라미터로 주어진 slotNo가 NIL인 경우 page의 모든 index entry들을 데이터 영역의 가장 앞부분부터 연속되게 저장하였다. 그 후, page header을 갱신하였다.

-edubtm_CompoactInteralPage()

이 함수는 Internal page의 데이터 영역의 모든 자유 영역이 연속된 하나의 continuous free area를 형성하도록 index entry들의 offset을 조정하였다. 먼저 파라미터로 주어진 slotNo가 NIL이 아니라면 slotNo에 대응하는 page의 모든 index entry들을 데이터 영역의 가장 앞부분부터 연속되게 저장하였다. 다음으로 파라미터로 주어진 slotNo가 NIL인 경우 page의 모든 index entry들을 데이터 영역의 가장 앞부분부터 연속되게 저장하였다. 그 후, page header을 갱신하였다.

-edubtm_Fetch()

이 함수는 파라미터로 주어진 page를 root page로 하는 index에서 첫 번째 object가 저장된 leaf index entry를 검색하고, 검색된 leaf index entry를 가리키는 cursor을 반환한다. 첫 번째 object는, 검색 조건을 만족하는 object들 중 검색 시작 key값과 가장 가까운 key값을 가지는 object를 의미한다. 먼저 파라미터로 주어진 root page가 internal page인 경우, 검색 조건을 만족하는 첫 번째 object가 저장된 leaf page를 찾기 위해 다음으로 방문할 자식 page를 결정한다. 그리고 결정된 자식 page를 root page로 하는 subtree에서 재귀적으로 edubtm_Fetch()를 호출하였다. 다음으로 파라미터로 주어진 root page가 leaf page인 경우, 검색 조건을 만족하는 첫 번째 object가 저장된 index entry를 검색하였다.

-edubtm_FetchNext()

이 함수는 검색 조건을 만족하는 현재 leaf index entry의 다음 leaf index entry를 검색하고, 검색된

leaf index entry를 가리키는 cursor을 반환한다. 검색 조건이 SM_GT, SM_GE일 경우, key값이 작아지는 방향으로 backward scan을 하고, 그 외의 경우 key값이 커지는 방향으로 forward scan을 한다.

-edubtm_FirstObject()

이 함수는 가장 작은 key값을 갖는 leaf index entry를 검색하는 함수이다.

-edubtm_LastObject()

이 함수는 가장 큰 key값을 갖는 leaf index entry를 검색하는 함수이다.

-edubtm_BinarySearchLeaf()

이 함수는 leaf page에서 파라미터로 주어진 key값보다 작거나 같은 key값을 갖는 index entry를 검색하고, 검색된 index entry의 위치를 반환한다.

-edubtm_BinarySearchInternal()

이 함수는 Internal page에서 파라미터로 주어진 key값보다 작거나 같은 key값을 갖는 index를 검색하고, 검색된 index entry의 위치를 반환한다.

-edubtm_KeyCompare()

이 함수는 파라미터로 주어진 두 key값의 대소를 비교하고 결과를 반환하는 함수이다. 이 때, Variable length string 의 경우에는 사전식 순서를 이용하여 비교한다.

3. 결론 및 토론

이번 프로젝트가 educosmos project 중 가장 어려웠던 project였다. 구현 초기에는 단지 data structure 을 구현하는 것 뿐이라고 생각하여서 간단할 것이라고 생각하였는데 세세한 부분까지

구현을 하다 보니 생각보다 많이 어려웠던 것 같다. 처음에 어려움에 봉착했던 부분은 free와 unused를 잘 이해하지 못해 발생하였다. Object의 insert와 delete가 발생할 때 free와 unused가 어떻게 변해야 하는지 이해하는데 꽤 오랜 시간이 걸렸다. 이를 완전히 이해한 후, 이를 실제적으로 구현하는데에도 오랜 시간이 걸렸다. 이를 구현하기 위해 printf를 통한 debugging으로 구현할 수 있었다. 다음으로 어려웠던 부분은 split함수를 구현하는 부분이었다. Split이 일어날 때 key의 범위를 설정하고 key를 page에 새로 삽입하는 부분에서도 어려움이 있었으며 키가 insert 될 때 생각처럼 정렬되지 않아 올바르게 정렬시키는데에도 오랜 시간이 걸렸다. 결국 이 모든 과정을 앞에서 언급한것과 같이 printf를 통한 debugging으로 해결할 수 있었다. 이번 프로젝트를 구현하는데 오랜 시간이 소요되었지만 그래도 모두 구현 완료하고 나니 뿌듯하였다. 이번 프로젝트에서는 key값의 중복을 허용하지 않는 범위에서 B+ tree를 구현하였는데, 다음에는 기회가 된다면 key값을 중복을 허용한 실제 우리가 DBMS에서 사용하는 B+ tree와 유사한 데이터 구조를 구현해보고 싶다.