Homework 4

파일처리

2013012148 이재일

# 과제 목표：

RM Layer API를 이용하여 작성한 연락처 프로그램에서 아래 항목들을 분석

1. Insert 기능과 Delete 기능이 이루어지는 동작순서

1. RM Layer API function 설명
2. Insert, Delete function 알고리즘 분석

**1. Insert 기능과 Delete 기능이 이루어지는 동작 순서**

1-1. Insert 기능 동작 순서

* 1. int main()에서 rmm.CreateFile("Contactlist.dat", recordSize) 호출
  2. int main()에서 rmm.OpenFile("Contactlist.dat", recordSize) 호출
  3. int main()에서 menuDisplay() 호출
  4. int main()에서 select변수에 menuDisplay()의 return값 저장
  5. int main()에서 switch(select) 사용해서 select가 1일 경우 case1 실행
  6. int insertContact(RM\_FileHandle &fh) 호출
  7. int main()에서 insertContact(RM\_FileHandle &fh) 호출
  8. int insertContact(RM\_FileHandle &fh)

6-1. 구조체 person 초기화

6-2. scanf를 통해 입력값을 받아서 구조체 person 내부 변수들에 각각 저장한다

6-3. char형 포인터 변수 record에 구조체 person의 주소값을 형변환을 한 후 저장한다

6-4. 인자로 받은 fh와 선언한 rid를 이용해 fh.InsertRec(record, rid)호출

1-2. Delete 기능 동작 순서

1. int main()에서 rmm.CreateFile("Contactlist.dat", recordSize) 호출
2. int main()에서 rmm.OpenFile(“Contactlist\_RM.dat”, fh)호출
3. int main()에서 select = menuDisplay()호출
4. int main()에서 switch(select)문 이후 case2에서 otherContact(fh, 2)호출
5. int otherContact(fh, 2)에서 char형 배열인 person\_name을 0으로 초기화 한 후 scanf로 입력 받은 입력 값을 person\_name에 저장한다
6. int otherContact(fh, 2)에서 rc = fs.OpenScan(fh, STRING, sizeof(char[32]), offsetof(PersonRecord, name), EQ\_OP, &person\_name, NO\_HINT) 호출한다
7. int otherContact(fh, 2)에서 for(rc = fs.GetNextRec(rec), n=0; rc == 0; rc = fs.GetNextRect(rec), n++) 을 선언한다
8. 선언한 for문 내부에서 val==2일 경우 rec.GetRid(rid)와 fh.DeleteRec(rid)를 실행한다.

# 2. RM Layer API function 설명

**# RM\_Manager**

1. CreateFile(FILE, recordSize)

: 1) PF\_Manager class통해 paged file을 생성한다

PF\_Manager의 CreateFile과 OpenFile을 호출해서 paged file을 얻는다

2) header page를 할당 후 해당 page pointer을 얻는다

PF\_FileHandle의 AllocatePage를 호출해서 header page 할당받는다. 해당 header page를 가리키는 pointer을 PF\_PageHandle의 GetData를 호출해서 얻는다.

3) file header를 작성한다(초기화한다)

RM\_FileHdr \*fileHdr에 GetData를 통해 얻은 header page를 가리키는 포인터를 형변환해서 대입한다. fileHdr의 firstfree는 RM\_RAGE\_LIST\_END로 초기화하여 처음에 빈 공간이 없음을 나타낸다. fileHdr의 recordSize는 CreateFile호출시 전달받은 recordSize를 집어넣는다. fileHdr의 numRecordsPerPage에는 전체 file size(PF\_PAGE\_SIZE이다)에 (RM\_PageHdr의 size + 1)을 뺀 값을 (recordsize+1.0/8)로 나눈 값을 넣어준다. 만약 (recordSize\*(fileHdr->numRecordsPerPage+1) + fileHdr->numRecordsPerPage/8)이 (PF\_PAGE\_SIZE – sizeof(RM\_PageHdr) -1)보다 크지 않으면 fileHdr->pageHeaderSize를 하나 더 증가시킨다. fileHdr의 pageHeaderSize에는 sizeof(RM\_PageHdr)와 bitmap의 크기인(fileHdr->numRecordsPerPage + 7) / 8 을 더한 값을 대입해준다. 마지막으로 fileHdr의 numRecords를 0으로 초기화한다.

1. OpenFile(FILE, fh) // fh의 private 변수에 file header info저장

: RM\_Manage의 맴버 필드인 PF\_Manager 포인터 변수 pPfm을 이용해서 PM\_Manager의 OpenFile을 호출한다. 호출 시 PF\_FileHandle의 포인터 변수 fh.pfFileHandle에 해당 paged file의 PF\_FileHandle이 저장된다. 이 fh.pfFileHandle의 GetFirstPage(pageHandle)을 호출해서 얻은 해당 page를 관리하는 pageHandle에서 GetData(pData)를 호출한다. 이 pData에 header information에 접근 할 수 있는 pointer가 저장된다. Memcpy(&fh.fileHdr, pData, sizeof(fh.fileHdr)를 호출해서 file header를 RM\_FileHandle의 buffer pool에서 read한다. 작업을 완료한 후 fh.pfFileHandle.UnpinPage를 호출해 header page를 unpin한다. fh.bHdrChanged에 FALSE를 대입해 file header의 가 변하지 않도록 설정해준다.

1. CloseFile(fh)

//fh가 다루는 file header에 변화가 생겼을 경우 file 갱신해준다

//open되어있는 file과 fh의 연결을 끊어 준다

: fh.bHdrChanged가 TRUE인 경우(file header에 변화가 있는 경우) file에 변화를 반영해주어야 한다. fh.pfFileHandle.GetFirstPage(pageHandle)을 호출해서 pageHandle이 header page을 get하도록 한다. pageHandle.GetData(pData)를 호출해서 pData에 header information에 접근할 수 있는 pointer를 얻는다. Memcpy(pData, &fh.fileHdr, sizeof(fh.fileHdr))를 호출해서 file header를 buffer pool에 write한다. 작업을 완료한 후 fh.pfFileHandle.MarkDirty(RM\_HEADER\_PAGE\_NUM)을 호출해서 header page를 dirty page로 mark해서 disk에서 해당 file을 갱신하도록 한다. Fh.pfFileHandle.UnpinPage(RM\_HEADER\_PAGE\_NUM)을 호출해서 header page를 unpin한다. 그리고 fh.bHdrChanged를 FALSE로 set해서 file header가 바뀌지 않도록 설정하는 것으로 갱신을 마친다. File을 close하기 위해서 pPfm->CloseFile(fh.pfFileHandle)을 호출한다. Memset(&fh.fileHdr, 0, sizeof(fh.fileHdr)를 호출하고 fh.fileHdr.firstFree를 RM\_PAGE\_LIST\_END로 초기화해주어서 member variables을 reset해준다.

4. DestroyFile(filename)

//이름이 filename인 RM file을 delete한다

: pPfm->DestroyFile(filename)을 호출해서 file delete한다.

# # RM\_FileHandle

5. GetRec(rid, rec)

//rid에 해당하는 record를 return한다

: rid.GetPageNum(pageNum)와 rid.GetSlotNum(slotNum)을 호출해서 rid로부터 page number와 slot number를 추출한다. slotNum이 0보다 같거나 크고, fileHdr.numRecordsperPage보다 작은지 bound check를 한다. Bound check를 통과하면 pfFileHandle.GetThisPage(pageNum, pageHandle)을 호출해서 rid가 가리켰던 pageNum에 해당하는 page를 얻는다. 해당 page에서 pageHandle.GetData(pData)를 통해서 pData에 data를 얻는다. GetBitmap(pData + sizeof(RM\_PageHdr), slotNum)을 호출해서 rid에 해당하는 record가 존재하는지 check하고 GetBitmap의 return값이 0이 나오면 존재함을 확인 할 수 있다. 인자로 전달받은 rec(RM\_Record)로 record를 복사하기 위해서 rec.rid에 rid를 대입한다. rec.recordSize에 fileHdr.recordSize 대입하고 rec.pData을 new char[rec.recordSize]를 통해 동적할당한 후 memcpy(rec.pData, pData + fileHdr.pageHeaderSize + slotNum + fileHdr.recordSize, fileHdr.recordSize)를 호출해서 복사를 완료한다. 작업을 완료한 후 pfFileHandle.UnpinPage(pageNum)을 호출해서 해당 page를 unpin해준다.

6. InsertRec(pData,rid)

//new record(pData)를 RF에 삽입하고 rid를 반환한다

: pData가 NULL인지 check한다(NULL일 경우 return). fileHdr.firstFree가 RM\_PAGE\_LIST\_END일 경우(free page list가 비어 있을 경우) pfFileHandle.AllocatePage(pageHandle)을 호출해서 new page를 할당받는다. pageHandle.GetPageNum(pageNum)을 호출해서 page number를 얻고, pageHandle.GetData(pData)를 호출해서 data pointer를 얻는다. ((RM\_PageHdr\*)pData)->nextFree에 RM\_PAGE\_LIST\_END를 넣어서 page header를 set한다. pfFileHandle.MarkDirty(pageNum)을 호출해서 next pointer에 변화가 있는 해당 page를 mark dirty해준다. 작업이 끝난 후 pfFileHandle.UnpinPage(pageNum)을 호출해서 page를 unpin한다. fileHdr.firstFree에 pageNum을 넣고 bHdrChanged에 TRUE를 넣는다. free page list가 비어있지 않다면(list에 사용 가능한 page가 있다면) pageNum에 fileHdr.firstFree를 집어넣는다. 얻은 pageNum으로 pfFileHandle.GetThisPage(pageNum, pageHandle)호출하고 pageHandle.GetData(pData)를 호출해서 pData에 data pointer를 저장한다. 이 pData로 for문을 0부터 page에 들어갈 수 있는 record 최대 개수까지 돌리는 과정에서 GetBitmap(pData + sizeof(RM\_PageHdr), slotNum)을 호출시켜서 empty slot을 찾는다. 해당 slotNum을 가지고 pRid = new RID(pageNum, slotNum)으로 rid를 assign하고 rid = \*pRid; delete pRid;를 실행하여 rid가 assign한 영역을 가리키게 한다. 그후 memcpy(pData + fileHdr.pageHeaderSize + slotNum\*fileHdr.recordSize, pRecordData, fileHdr.recordSize)를 호출하여서 주어진 record data를 buffer pool로 복사한다. 해당 slot위치에 record가 삽입되었으므로SetBitmap(pData + sizeof(RM\_PageHdr), slotNum)을 호출해서 bit를 set해준다. 작업을 완료한 후 slotNum을 0부터 numRecordsPerPage까지 순회하여 빈 slot이 없을 경우 free page에서 해당 page를 제거한다. pfFileHandle.MarkDirty(pageNum)과 pfFileHandle.UnpinPage(pageNum)을 호출하여 header page를 dirty page로 mark하고 unpin해준다.

7. DeleteRec(rid)

//rid에 해당하는 record를 삭제한다

: rid.GetPageNum(pageNum)과 rid.GetSlotNum(slotNum)을 호출하여 rid의 pageNum과 slotNum을 추출한다. 얻은 slotNum은 bound check를 한번 해준다. pageNum을 pageHandle에 인자로 전달하고 pageHandle.GetData(pData)에서 pData에 data를 얻는다. GetBitmap(pData + sizeof(RM\_PageHdr), slotNum)을 호출해서 해당 slotNum에 record가 존재하는지 check해준다. 해당 slotNum에 record가 존재하면 ClrBitmap(pData + sizeof(RM\_PageHdr), slotNum)을 호출해서 해당 slotNum의 bit를 clear해준다. 그 후 memset을 호출해서 해당 slotNum에 해당하는 record를 NULL로 초기화해준다. 해당 page의 bitmap을 검사해서 만약 빈 slot이 없다면 해당 page를 freelist에서 제거한다. 빈 slot이 있다면, freelist에 넣어준다. 해당 page를 mark dirty하고 unpinPage함으로 delete를 마무리한다.

8. UpdateRec(rec)

//해당 record를 update한다

: rec.GetRid(rid)를 호출해서 rid를 얻는다. rec.GetData(pRecordData)를 호출해서 record data를 얻는다. rid.GetPageNum(pageNum)과 rid.GetSlotNum(slotNum)을 호출해서 rid로부터 pageNum과 slotNum을 추출한다. slotNum은 bound check를 해준다. rec.recordsize와 fileHdr.recordsize를 비교해서 updating record의 recordsize와 file handle이 일치하는지 확인한다. pageNum으로 pfFileHandle.GetThisPage(pageNum, pageHandle)와 pageHandle.GetData(pData)를 호출해서 해당 page pointer를 pData에 얻는다. pData를 통해 해당 page에 접근해서 GetBitmap으로 해당 slotNum에 record가 존재하는지 check한다. record가 존재할 경우 memcpy(pData + fileHdr.pageHeaderSize + slotNum \* fileHdr.recordSize, pRecordData, fileHdr.recordSize)를 호출하여서 해당 record를 update한다. header page를 MarkDirty하고 UnpinPage함으로 작업을 종료한다.

9. ForcePages(pageNum)

//page를 buffer pool로부터 disk에 넣는다.

: bHdrChanged를 통해 해당 file의 header에 변화가 있었을 경우 header page의 위치를 가리키는 pData에 pointer를 얻어서 memcpy(pData, &fileHdr, sizeof(fileHdr))를 호출해서 buffer pool에 file header를 쓴다. header page를 MarkDirty하고 UnpinPage한 후 ForcePage한다. bHdrChange를 FALSE를 집어넣어서 file header가 바뀌지 않도록 설정한다. 그 후 pfFileHandle.ForcePages(pageNum) 호출함으로 ForcePage를 완료한다.

10. GetBitmap(map, idx)

//주어진 idx에 해당하는 bit을 return한다

: (map[idx/8] & (1 << (idx%8))) !=0을 return한다

11. SetBitmap(map, idx)

//주어진 idx에 해당하는 bit를 set한다

: map[idx / 8] |= (1 << (idx % 8));을 실행한다(or연산)

12. ClrBitmap(map, idx)

//주어진 idx에 해당하는 bit를 clear한다

: map[idx / 8] &= ~(1 << (idx & 8));을 실행한다(and 연산)

# # RM\_Record

13. GetData(\_pData)

//data를 얻는다

: pData가 NULL인지 확인한다. NULL이 아닐 경우 \_pData에 pData를 대입한다

14. GetRid(\_rid)

//rid를 얻는다

: pData가 NULL인지 확인한다. NULL이 아닐 경우 \_rid에 rid를 대입한다.

15. Set(pData2, size, rid\_)

//record의 rid와 data를 set한다

: recordSize에 size를 this->rid에 rid\_를 대입한다. pData가 NULL일 경우 pData에 recordSize크기의 char배열을 할당한다. memcpy(pData, pData2, size);를 호출해서 pData에 pData2를 setting한다.

# # RM\_FileScan

16. OpenScan(fh, \_attrType, \_attrLength, \_attrOffset, \_compOp, \*\_value, \_pinHiny)

//fh와 scan condition으로 file scan을 open한다

: bScanOpen이 false이고, fh.fileHdr.recordSize가 0이 아닌지 check한다. \_compOp, \_attrType, \_attrLength, \_attrOffset 모두 sanity check를 한다. 이상이 없을 경우 local variable에 parameter들을 copy한다. bScanOpen에 TRUE 대입하고, curPageNum에 RM\_HEADER\_PAGE\_NUM 대입하고 curSlotNum에 pFileHandle->fileHdr.numRecordsPerPage를 대입한다.

17. GetNextRec(rec)

// scan condition을 만족시키는 next record의 copy를 회수한다

: bScanOpen과 pFileHandle->fileHdr.recordSize를 check한다. curSlotNum과 numRecordsPerPage가 같을 경우 GetNextPage(curPageNum, pageHandle)과 pageHandle.GetPageNum(curPageNum)을 호출하여 다음 page를 가져온다.curPageNum을 이용하여 GetThisPage(curPageNum, pageHandle)을 호출하고 pageHandle.GetData(pData)를 호출하여 해당 page의 header에 접근할 수 있는 pointer를 pData에 얻는다. FindNextRecInCurPage(pData)를 호출하여서 scan condition에 따른 next record를 찾는다. 해당 페이지에서 next record를 찾지 못할 경우 다음 page에서 찾는다. next record를 찾은 경우 given location에 record를 copy한다. curSlotNum을 1 증가시킨다. 그 후 해당 page에서 더 이상 matching되는 record가 없을 경우 FindNextRecInCurPage(pData)을 호출하여 GetNextRec()을 호출할 필요가 없도록 한다. 그후 UnpinPage(curPageNum)하고 마무리한다.

18. FindNextRecInCurPage(pData)

// 현재 page에서 hit 혹은 end 할 때 까지 slot을 반복해서 순회한다

: curSlotNum부터 numRecordsPerPage까지 for문을 선언하여 next record를 반복해서 찾는다. for문 내부에 GetBitmap으로 empty slot을 skip하고 compOp가 NO\_OP일 경우 hit이므로 빠져나온다. attrType에 따라 cmp에 구별되는 값을 대입한다. 그후 compOp와 cmp을 같이 비교하여서 hit여부를 판단한다.

19. CloseScan()

//file scan을 close한다

: bScanOpen가 open인지 check한다. open일 경우 member variables을 초기화한다.

# 3. Insert, Delete function 알고리즘 분석

3-1. insertContact

1. local variable 로 PersonRecord person을 선언한다.

2. person 변수를 0으로 초기화한다.

3. scanf함수를 통해서 person의 맴버변수 name, addr, telnum, email에 각각 입력값을 입력한다.

4. 미리 선언한 char형 포인터 변수 record에 person변수의 주솟값을 char형 포인터 변수로 형변환하여 대입한다.

5. 인자로 전달받은 RM\_FileHandle 참조자 fh로 fh.InsertRec(record, rid)를 호출한다.

6. InsertRec함수 내부에서 firstFree에 free page가 있는지 check하고 없는 경우 새 page를 할당해준다. free page가 있는 경우 firstFree에서 pageNum을 얻는다. pageNum에 해당하는 page의 free slot를 찾고, 해당 slot의 bitmap bit를 set하고, memcpy를 이용해서 record를 삽입한다. 삽입을 완료한 후 record를 삽입한 해당 page가 빈 slot이 없을 경우 freelist에서 제거한다. MarkDirty와 UnpinPage를 한 후 작업을 마무리 한다.

3-2. othersContact (Delete)

1. RM\_FileScan의 OpenScan 함수를 통해 삭제하려는 record를 찾는다. OpenScan 내부적으로는 인자들이 적절한지 체크한 뒤 인자들을 RM\_F ileScan의 지역 변수에 각각 저장한다. 그리고 RM\_FileScan의 상태 를 초기화한다. 여기서 저장한 값들을 토대로 GetNextRec에서 Recor d를 탐색한다.

2. for문을 돌면서 원하는 record를 찾는다. for문은 fs.GetNextRec(rec)로 next record를 검색하고 fs.GetNextRect(rec)의 return값이 0이 아닐 때까지 반복한다.

3. for문 내부에서 rec.GetRid(rid)을 호출하여 rec의 record id를 rid에 저장한다. 그리고 fh.DeleteRec(rid)를 호출한다.

4. fh.DeleteRec(rid)를 호출하면 rid에서 pageNum과 slotNum을 추출해서 해당 pageNum과 slotNum에 record가 존재하는지 GetBitmap함수를 통해서 확인한다. record가 존재하지 않을 경우 return하고 for문을 돌면서 다음 record를 찾는다. record가 존재할 경우 ClrBitmap함수로 bitmap에 해당 record slot bit을 clear해주고 해당 slot을 NULL로 memset해준다. 해당 record를 찾은 page가 빈 slot이 없다면 freelist에서 제거하고 빈 slot이 있다면 freelist에 추가해준다. 마지막으로 MarkDirty와 UnpinPage로 header page를 dirty로 mark하고 unpin해준다. 그리고 return 0을 해주어 for문을 빠져나온다.