2022-01 데이타베이스 시스템 설계과제 프로젝트 중간 보고서

01분반 20172848 정석우

1. 설계 개요

가변 길이 레코드 포맷과 Slotted Page Structure에 기반한 Relational Database System 설계 및 TDD를 적용한 구현

2. 요구 사항 Spec

2-1. 응용으로의 요구사항

1) 시스템은 CLI 기반 Application으로써 동작해야 한다.

2) 시스템은 다른 응용에서 데이터 저장/조회를 위한 API를 제공해야 한다.

2-2. 데이터 삽입 요구사항

1) 시스템의 데이터는 파일에 저장되어야 하고, 시스템이 재시작/종료/초기화 되어도 파일에 있는 데이터의 무결성 (Persistence)는 보장되어야 한다.

2) 시스템 내부에서 테이블 생성 및 레코드 삽입이 가능해야 한다.

3) 시스템에 입력되는 데이터는 VARCHAR(N), (실제 프로그램 상에는 String) 만을 대상으로 한다.

2-3. 데이터 검색 요구사항

1) 시스템 내부에서 테이블의 PK 값으로 레코드 검색이 가능해야 한다.

2) 시스템 내부에서 테이블의 컬럼 목록을 조회할 수 있어야 한다.

3. 시스템 설계

3-1. 전체 시스템 도식도

3-1-1. 가변 길이 레코드 포맷

null bitmap	A, 고정 길이	B, 고정 길이	C, 가변 길이	D, 가변 길이	C data	D data

칼럼이 4개 있는 테이블의 레코드를 예시로 들어 도식화하였다.

1. null bitmap은 가변 길이 레코드의 맨 앞에 위치한다. 총 4 Byte로, 컬럼의 최대 개수는 32개를 지원한다.

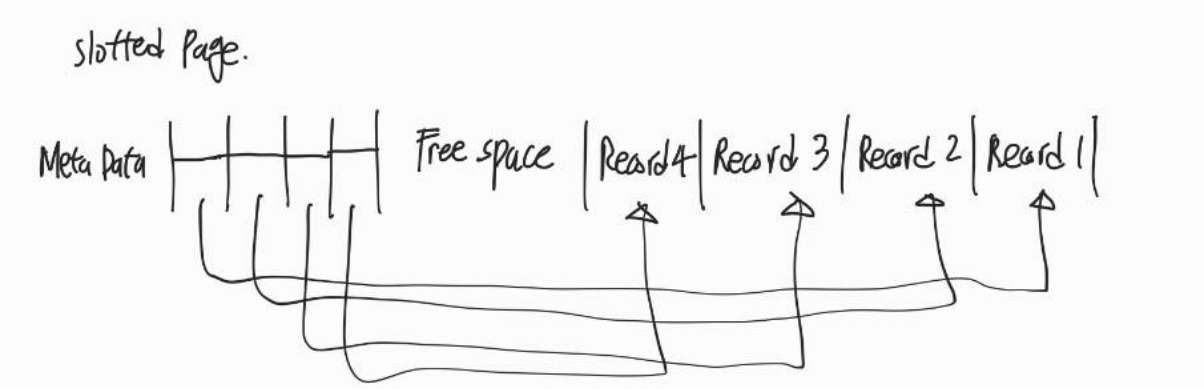
2. 고정 길이 컬럼의 경우, 최대 길이인 MAX_COLUMN_LEN을 50 Byte로 설정한다. 따라서 고정 길이 칼럼은 최대 50자의 데이터를 저장할 수 있다.

3. 가변 길이 칼럼의 경우, 데이터가 저장되어 있는 위치를 읽어올 때 (byte, offset) 데이터 쌍을 사용하여 접근한다. 아래는 구조체로 구성한 byte, offset 데이터 쌍 예시이다.

```
struct variable_column_loc {
    int byte;
    int offset;
};
```

가변 길이 레코드에는 특별한 메타 데이터를 저장하지 않는다. 테이블에 종속되는 특성 데이터는 아래에 기술할 slotted page의 메타 데이터에 저장한다.

3-1-2. Slotted Page Structure



Slotted Page Structure의 메타데이터는 테이블에 종속되는 몇 가지 정보를 포함한다.

1. Page 전체 크기는 4KB (4096 Byte)로 고정한다.

2. 현재 Page에 존재하는 레코드 entry 개수

3. 현재 Page의 Free Space 끝 주소

메타 데이터는 구조체로 구성되며, 메타 데이터 이후에는 각 레코드들이 어디에 저장되어 있는지 가르키는 포인터 배열이 존재한다. Page 전체 크기가 4KB이기 때문에 가변 길이 레코드들이 채워지지 않은 공간은 Free Space로 남는다.

3-1-3. System Catalog

System Catalog (Data Dictionary)에 포함되는 정보는 다음과 같다.

1. 테이블의 이름
2. 테이블의 각 컬럼 이름
3. 테이블의 각 컬럼 유형 (가변 길이인지, 고정 길이인지, pk인지)
4. 테이블의 가변 길이 컬럼 개수
5. 테이블의 고정 길이 컬럼 개수
6. 테이블이 어느 파일의 어느 블록에 나누어져 위치하는지
7. 다음 삽입 시 레코드는 어느 파일의 어느 블록에 저장할 지

이 System Catalog는 JSON Array 형태로 파일에 저장되며, 시스템이 초기 가동될 때 파일에서부터 읽어와 메모리에 적재한다. 아래는 Student 테이블의 간단한 System Catalog 예시이다.

{

table_name : “student”,

table_column_list : [“id”, “name”, “age”, “major”],

is_column_variable : [false, true, false, true],

variable_column_cnt : 2

fixed_column_cnt : 2

block_location :

[

{ file : “A”, start_loc : 4096, end_loc: 49152},

{ file : “C”, start_loc : 0, end_loc: 4096}

],

next_insert_location : { file : “C”, start_loc : 0, end_loc : 4096}

}

테이블에 대한 메타데이터 이외에 시스템 운영에 필요한, 파일에 들어갈 수 있는 Block의 최대 개수 같은 전반적인 데이터 또한 동일한 json 파일에 저장한다. 가령 새 블럭 삽입을 하는 경우 다음 블럭이 삽입되는 위치를 메타데이터에 저장해 놓고, 이후 데이터가 삽입된 이후에 이를 업데이트 한다. 해당 데이터를 json 형식으로 표현해보면 다음과 같다.

{

FILE_MAX_BLOCK_NUM : 1024,

NEXT_BLOCK_LOC : { file : “D”, loc : 0}

}

3-2. Pseudo code

3-2-1. 테이블 생성 Pseudo code

System Catalog에 새 json object를 삽입한다.

다음 레코드가 삽입될 위치를 읽어온다

3-2-2. 레코드 삽입 Pseudo code

입력 받은 데이터로 가변 길이 레코드 구조체를 생성.

새로 생성된 가변 길이 레코드를 System Catalog로부터 어디에 삽입할 지 읽기.

해당 파일의 Block을 읽기.

레코드 포인터 배열 끝에 새 레코드의 위치를 가리키는 새 포인터 삽입

읽어온 Block의 Free space 마지막 주소 읽기

삽입되는 레코드의 위치를 계산. (시작 주소 : Free Space 마지막 주소 – (Block의 크기) 끝 주소 : 기존 Free space 마지막 주소)

새 레코드를 해당 위치에 삽입

레코드 포인터가 새로 삽입된 레코드를 가리키도록 갱신

System Catalog의 다음 레코드 삽입 위치와 레코드 저장 위치 갱신

3-2-3. 레코드 검색 Pseudo code

한 레코드를 특정 테이블에서 PK 값으로 검색하는 경우의 예시. (SELECT * FROM Student WHERE id=12)

Student 테이블의 레코드들이 어떤 파일의 어떤 Page에 저장되어 있는지 확인

for 레코드들이 저장된 모든 파일들:

    해당 파일의 block들 순차적으로 읽기.

    block의 레코드들에 접근

    if (접근한 레코드의 id=12이면):

        return Record

3-2-4. 컬럼 검색 Pseudo code

System Catalog로부터 테이블의 메타데이터 읽기

메타데이터로부터 컬럼 데이터 출력

4. 세부 구현 사항

C++ 20을 사용하여 구현할 예정이다.

파일의 일부분만 읽고 / 쓰는 용도로 C++ iostream 헤더의 seekp, seekg 함수를 사용할 예정이다.