20176342 송민준

관계 DB 저장 시스템 개발 최종 보고서

Meta-data의 Meta-data 설계

저장시스템 home directory에 meta.meta 파일 생성 후 각 table.meta 파일의 위치와 개수 등의 정보를 가지게 설계합니다.

아래와 같이 3개의 테이블이 있는 DBMS 시스템에서는 3개의 메타파일이 생성되고 각각의 디렉토리 경로가 meta.meta 파일에 저장됩니다.

Metafile [2022-05-06 00:00:01] 생성일자 기록

student.meta ./student/ (메타 파일명, 폴더 경로)
player.meta ./player/
university.meta ./university/

~~3 meta files (메타 파일 개수)~~

변경 사항 : 메타파일에 메타파일의 개수를 기록할 필요가 없어서 기록하지 않게 변경했습니다.

Meta-data의 Meta-data 구현

```
def createTable(tableName, args):

    # 메타메타 파일 열자

    try:
        meta = open("./meta.meta", 'r+')
    except:
        meta = open("./meta.meta", "w")
        meta.writelines("JOON DB meta.meta [" + datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "]")
        meta.writelines("\n")
        meta.writelines("\n")
        meta.flush()
        meta.close()
        meta = open("./meta.meta", 'r+')
```

위 첨부한 소스코드에서는 테이블 생성 질의를 입력 받을 시, meta.meta 파일이 없으면 현재 날짜와 함께 생성하게 구현하였습니다.

```
while True:
    readData = meta.readline()

    if readData != "":
        currTableName = readData.split(".meta")[0]
        if tableName == currTableName:
            err("tableName already exists")
    else:
        break

meta.writelines("\n" + tableName + ".meta ./" + tableName + "/")
meta.flush()
meta.close()
```

테이블 생성시 meta.meta를 처음부터 끝까지 읽어서 테이블명이 이미 존재하는 지 체크하는 로직을 추가했습니다. 만약 테이블명이 존재하지 않는다면 테이블명.meta을 meta.meta에 기록하게 됩니다.

meta.meta

```
1   JOON DB meta.meta [2022-05-25 18:03:46]
2
3
4   student.meta ./student/
```

위는 student 테이블을 생성하는 질의를 실행한 후 meta.meta 파일의 내용입니다.

메타 파일 설계

메타 파일은 테이블명.meta 파일로 각 테이블명 디렉토리에 저장되며,

메타 파일의 내용은 아래와 같이 테이블명, 생성일자, 컬럼과 컬럼의 형태(CHAR, VARCHAR), 그리고 해당 디렉토리에서 사용하는 슬롯들의 파일명(슬롯단위 파일I/O)이 저장되어 있고,

총 슬롯 페이지 수와 레코드 수가 저장되어 있습니다.

Ex) student 테이블 meta 파일 (student.meta)

파일 내용물

student.meta [2022-05-06 00:00:01] (생성일자 기록)

id CHAR(10)
name VARCHAR(20)
grade CHAR(3)

~~**0001.slot**~~
~~**0002.slot**~~
~~**0003.slot**~~

~~3 SLOTS (총 슬롯 페이지 수)~~
~~120 ROWS (총 레코드 수)~~

변경 사항 : 테이블의 메타 파일에 슬롯 번호와 슬롯 페이지 수, 레코드 수를 기록하지 않게 변경했습니다. 해당 내용들이 없어도 기능 구현을 하는 데 문제가 없었습니다.

메타 파일 구현

```
# 테이블 폴더 만들자
createDirectory("./" + tableName)
# 메타 만들자

meta = open("./" + tableName + "/" + tableName + ".meta", "a+")
meta.writelines(tableName + ".meta [" + datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "]")
meta.writelines("\n")
meta.writelines("\n")

for arg in args:
    meta.writelines("\n" + arg[0] + " " + arg[1])

return tableName+" is created successfully."
```

테이블 생성시 createDirectory함수에 의해서 테이블명으로 폴더를 생성하고,

테이블명/테이블명.meta 파일을 생성하게 됩니다.

그 후 테이블명.meta파일에 컬럼들의 정보에 대해서 기록합니다.

예를 들어 'CREATE TABLE student(name CHAR(10) grade CHAR(3) department VARCHAR(20) introduction VARCHAR(50))' 라는 student 테이블 생성 질의를 실행할 경우,

student > student.meta

```
1   student.meta [2022-05-26 08:03:06]
2
3
4   name CHAR(10)
5   grade CHAR(3)
6   department VARCHAR(20)
7   introduction VARCHAR(50)
```

위와 같은 student.meta 파일을 생성하게 됩니다.

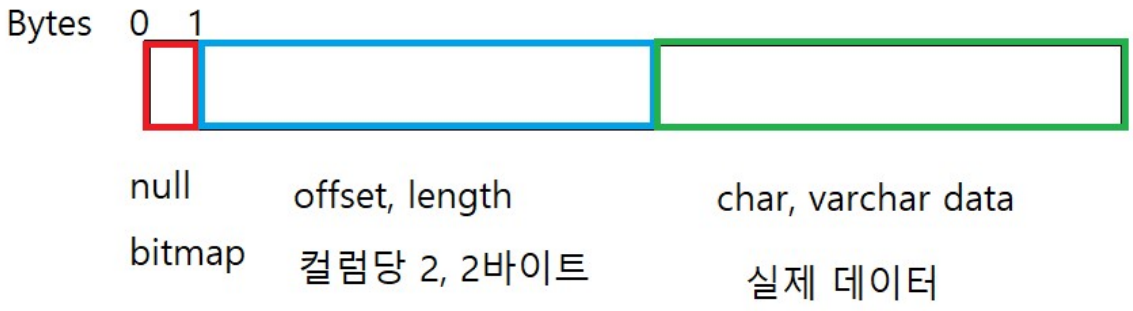
파일 구조 설계

저장시스템 home directory 기준으로, 테이블당 테이블명으로 폴더 생성. 각 폴더에는 slotted page의 개수 등 슬롯 페이지 관련 데이터를 담고 있는 table.meta 파일 존재.

각 슬롯 페이지 헤더에서는 레코드가 어디 있는지에 대한 Pointer를 담고 있게 설계합니다.

변경사항 : 테이블의 메타 파일에는 슬롯 개수 등 관련 정보를 담고 있지 않습니다.

가변길이 레코드 포맷



Null bitmap을 가변길이 레코드 제일 앞으로 오게 설계합니다.

Null bitmap은 8비트(1바이트)로 한 레코드에서 최대 컬럼을 8개만 가질 수 있게 설계합니다. Offset(2bytes) , Length(2bytes) 로 설계해서 총 Length(99)*컬럼 개수(8) 대략 레코드 하나의 크기는 1000바이트가 최대가 되게 합니다. 한 슬롯 페이지는 4000바이트가 최대가 되게 구현을 할 것이므로 슬롯 하나에 레코드가 아무리 적게 들어가도 4개의 레코드가 들어가게 됩니다.

최소는 컬럼 1개*length(1) + null bitmap 1바이트 => 3바이트가 되게 합니다. 이 경우에는 슬롯 하나에 레코드가 대략 1333개 정도 들어갈 수 있습니다. (4000/3=1333.333...)

Null bitmap이 1이 될 시 해당 컬럼 Byte 데이터 회수 설계합니다.

가변길이 레코드 포맷 구현

```
def insertRow(tableName, args):

    # 메타에 쓰이는 3종 세트
    cols = []
    types = []
    sizes = []
    colData = []
    # 우선 테이블 메타 파일 열어야됨

    try:
        reader = open("./" + tableName + "/" + tableName + ".meta", "r")
    except:
        err(tableName + " meta not exists")

    while True:
        readData = reader.readline()

        if " CHAR(" in readData or " VARCHAR(" in readData:
            cols.append(readData.split(' ')[0])
            types.append(readData.split(' ')[1].split('(')[0])
            sizes.append(int(readData.split(' ')[1].split('(')[1].split(')')[0]))

            temp = {}
            temp["cols"] = readData.split(' ')[0]
            temp["types"] = readData.split(' ')[1].split('(')[0]
            temp["sizes"] = int(readData.split(' ')[1].split('(')[1].split(')')[0])

            colData.append(temp)
        if readData == "":
            break
```

'INSERT INTO student VALUES("minjoon","A+","CSE","hello world Nice to meet you")'

만약에 사용자로부터 위의 INSERT 질의를 받았다고 가정했을 시, 먼저 student 메타 파일을 열어서 student 테이블의 컬럼들이 어떤 속성들을 가지고 있는 지 불러와야 됩니다. 위의 첨부한 소스 코드가 insertRow함수인데, 메타파일을 먼저 open하고, 끝까지 읽어서 컬럼들의 정보를 읽게 됩니다.

args는 minjoon, A+, CSE, hello world Nice to meet you가 됩니다.

```
for i in range(len(args)):
    #empty string or null is preprocessed
    #convert to variable length record
    arg = args[i]

    if len(arg) > sizes[i]:
        err("column "+cols[i]+" exceed length limit "+str(sizes[i])+"  ("+str(len(arg))+")")

    if arg is None or arg == "NULL":
        nullBitMap += "1"
        arg = ''
        args[i] = ''
    else:
        nullBitMap += "0"

    if types[i] == "CHAR":
        record += rpad(arg,sizes[i],' ')
    elif types[i] == "VARCHAR":
        record += str(varcharOffset)+lpad(str(len(arg)),2,'0')
        varcharOffset += len(arg)
```

```
varcharOffset += len(list(filter(lambda x: x == "VARCHAR", types))) * 4
varcharOffset += sum( [x["sizes"] for x in colData if x["types"] == "CHAR"] )
```

레코드에 가변길이 컬럼의 오프셋과 사이즈를 기록하기 위해 varcharOffset이라는 변수를 만들어서 varchar가 시작되는 위치를 계산했습니다.

```
    #이제 슬롯을 찾아서 레코드를 넣을거임

    slotCurr = 0
    while True:
        if insertSlot(tableName, slotCurr, record) == 1: #성공했을 시 루프 빠져나간다..
            break
        slotCurr += 1

    return tableName+ " inserted 1 row successfully."
```

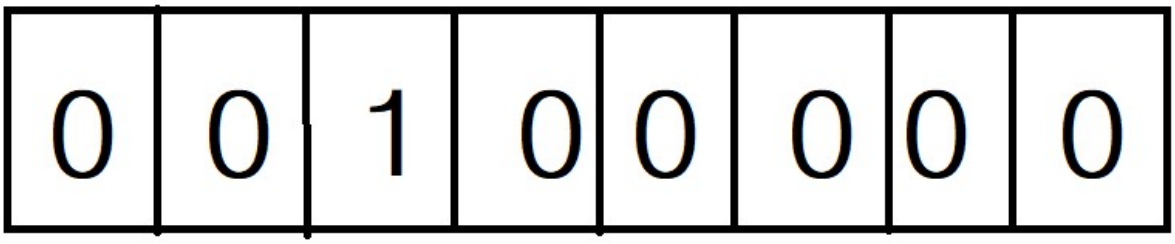
그 후 레코드를 슬롯에 삽입해야 되는데 0번 슬롯부터 빈 공간이 있는지 확인해서 성공할 때 까지 슬롯 번호를 증가하는 식으로 구현했습니다.

Null Bitmap 설계 및 구현

```
for i in range(len(args)):
    #empty string or null is preprocessed
    #convert to variable length record
    arg = args[i]

    if len(arg) > sizes[i]:
        err("column "+cols[i]+" exceed ler

    if arg is None or arg == "NULL":
        nullBitMap += "1"
        arg = ''
        args[i] = ''
    else:
        nullBitMap += "0"

    if types[i] == "CHAR":
        record += rpad(arg,sizes[i],' ')
    elif types[i] == "VARCHAR":
        record += str(varcharOffset)+lpad(
        varcharOffset += len(arg)

for i in range(len(args)):
    arg = args[i]
    if types[i] == "VARCHAR":
        record += arg

# args = ["minjoon", "A+", "hello world th
for i in range(len(nullBitMap)):
    if nullBitMap[i] == "0":
        nullBitMapArr.append(0)
    else:
        nullBitMapArr.append(1)


nullBitMap = rpad(nullBitMap,8,'0')
```

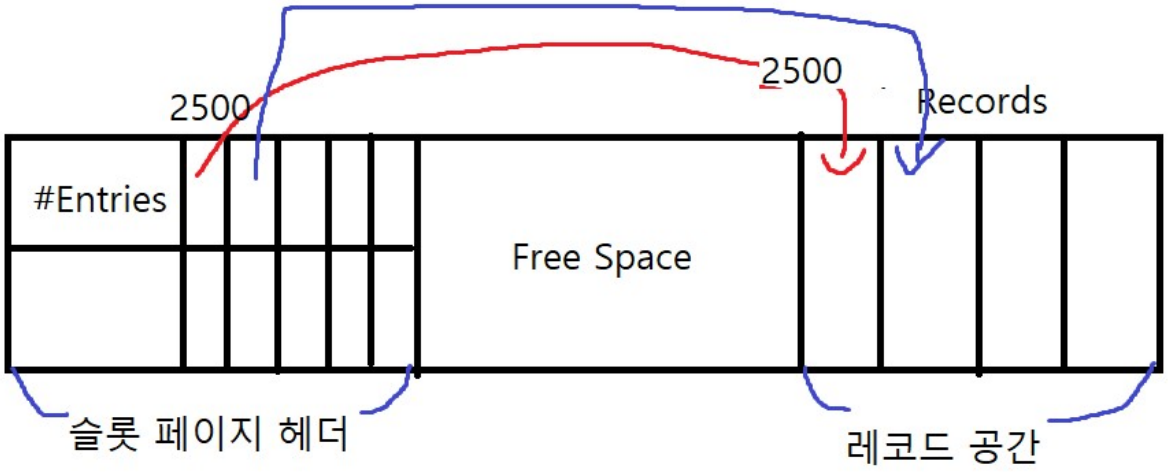
INSERT 질의에서 들어온 파라미터의 값이 NULL이면, nullBitMap에 1을 추가해주고 그게 아니면 0을 추가했습니다. 그리고 컬럼의 개수가 8개가 안된다면 8자리가 될때까지 0을 채워줬습니다.

예를 들어 컬럼이 4개인 테이블에 3번째 컬럼을 NULL로 한다면,



Null bitmap 이 위와 같이 되고, 여기에 +1을 해줍니다. (바이너리에서 0은 NULL로 인식되기 때문에 임의로 1을 더해줬습니다.) 따라서 00100001이 되고, 10진수로 33 이 됩니다.

Slotted page 구조



슬롯 페이지 크기는 한 슬롯당 4000바이트로 설계합니다.

Free space 구현은 교재와 같이 슬롯 페이지의 중간 부분이 Free Space가 될 수 있도록 설계, #Entries 부분은 1바이트로 할당하여 슬롯 페이지 1개당 2^8-1개의 레코드를 최대로 가질 수 있게 설계,

레코드 포인터, End of Free Space 포인터는 2바이트씩 할당하여 4000바이트의 블록 크기를 처음부터 끝까지 가리킬 수 있게 설계,

레코드 포인터가 2바이트인 이유는 1바이트로는 2^8=64 바이트 까지만 가리킬 수 있기 때문에

2바이트 = 2^16 = 65536-1 바이트 까지 가리킬 수 있게 2바이트로 설계했습니다.

Slotted page 구조 구현

위의 가변길이 레코드 구현에서 나온 레코드 값을 슬롯에 삽입할 때 insertSlot()함수를 사용합니다.

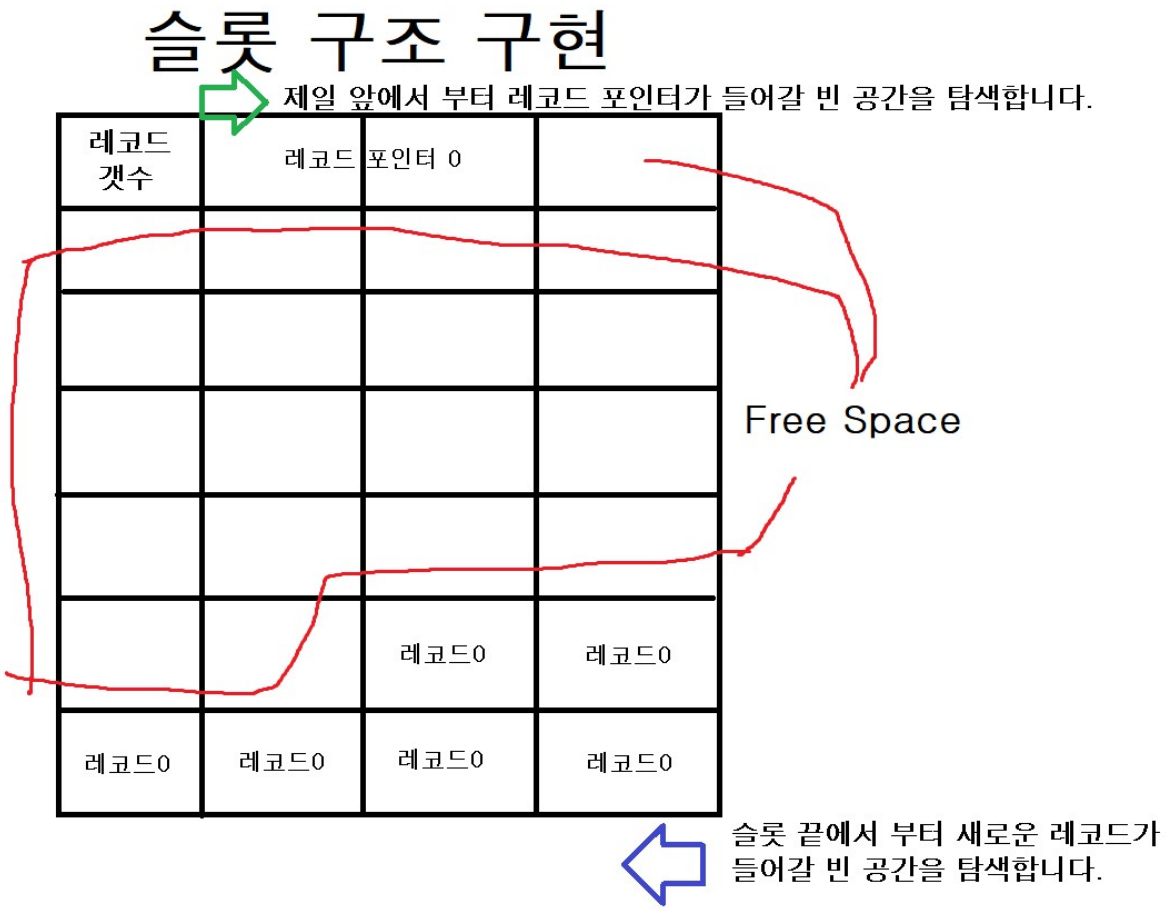
```
#특정 슬롯에 레코드를 삽입하는 함수, 공간이 부족해서 실패하면 -1리턴
def insertSlot(tableName, slotNum, record):
    if checkSlot(tableName, slotNum) == False:
        #없으면 만들자
        createSlot(tableName, slotNum)

    try:
        slot = open("./" + tableName + "/slot" + lpad(slotNum,3,'0') + ".bin", "rb+")
    except:
        err("table "+ tableName+ " slot "+slotNum +" not exists ")

    #빈공간 찾아야됨

    slot.seek(0)
    count = slot.read(1)
    count = int(struct.unpack('b',count)[0])


    emptyPointer = -1
    slot.seek(SLOT_SIZE) ##마지막으로 커서를 옮겨서 한 바이트씩 읽어서 \0이 아닐때까지 읽는다

    for i in range(0,SLOT_SIZE-len(record)-2-1-count*2): #레코드 포인터가 2바이트라서 2를 추가적으로 뺐다( 레코드 포인터의 여유 공간까지 고려 )
        slot.seek(SLOT_SIZE-i)

        readData = slot.read(1)

        if readData == b'\x00':
            emptyPointer = SLOT_SIZE-i
            if i != 0:
                emptyPointer += 1
            break

    if emptyPointer == -1:
        slot.close()
        return -1  # 해당 슬롯에 빈공간이 없어서 -1리턴
```

먼저 checkSlot()함수를 통해 해당 슬롯이 존재하는 지 체크하고, 없으면 createSlot()함수를 통해 만들어 줍니다. 그 후 슬롯의 맨 뒤 바이트부터 체크해서, 비어있는 공간의 address를 찾습니다.

```
startPoint = emptyPointer-len(record)
slot.seek(startPoint)
slot.write(record.encode())

slot.seek(0)

recordPointer = -1
i = 1
while i < SLOT_SIZE-1:
    slot.seek(i)
    readData = slot.read(2)
    data = int(struct.unpack("H",readData)[0])

    if data == 0:
        recordPointer = i
        break
    i += 2


if recordPointer == -1:
    err("슬롯 공간은 있는데 레코드 포인터가 없는 말도 안되는 경우")
slot.seek(recordPointer)
slot.write( struct.pack('H',startPoint)  )
```

따라서 emptyPointer는 슬롯의 맨 뒤부터 탐색해서 첫번째로 발견한 빈 공간의 주소가 됩니다. 슬롯 하나의 크기가 4000바이트 이므로 레코드를 처음 삽입할 때 emptyPointer는 4000이 됩니다. 그런데 레코드 포인터는 레코드의 끝을 가리키는 게 아니라 시작을 가리켜야 되므로 emptyPointer – record의 길이를 가리켜야 됩니다. 따라서 startPoint는 레코드의 시작 포인터가 됩니다.

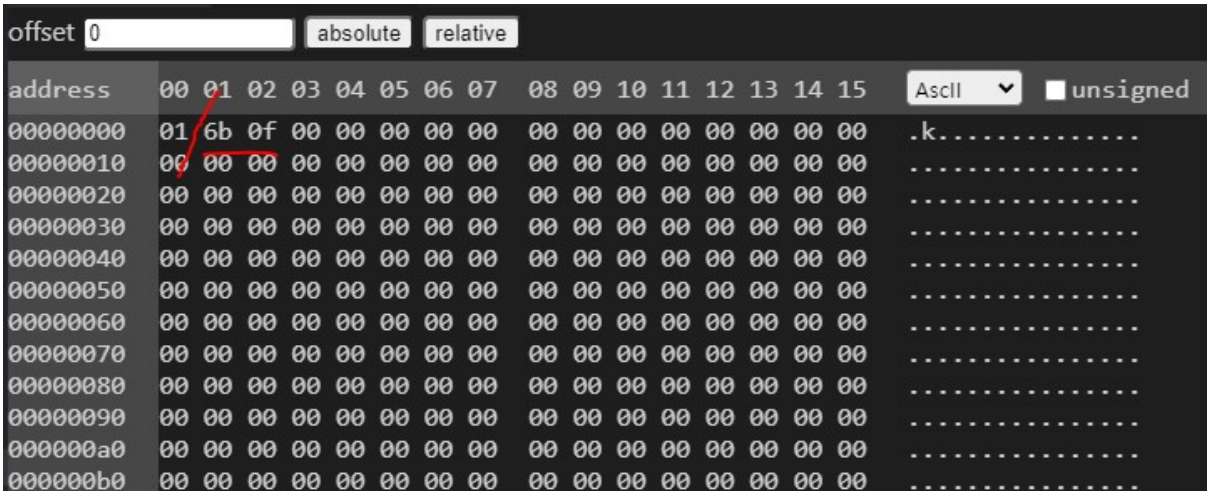
그 후 레코드 포인터를 기록할 위치를 찾아야 되는데, 슬롯의 처음부터 탐색해서 빈 공간이 나온다면 그 장소에 2바이트의 크기로 startPointer의 값을 기록해줍니다.



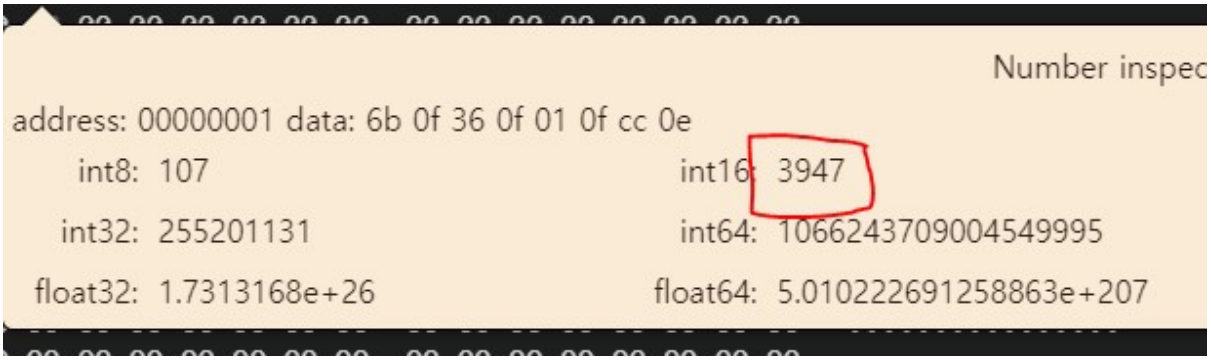
따라서 위와 같은 그림으로 공간을 탐색해서 Free Space를 사용해 나가는 구조가 됩니다.

'INSERT INTO student VALUES("minjoon","A+","CSE","hello world Nice to meet you")'

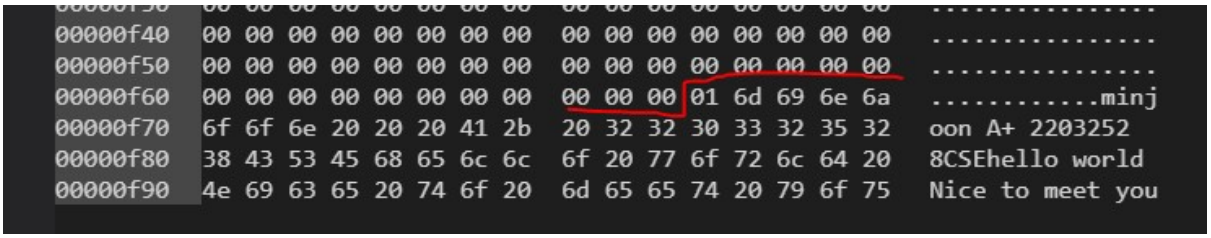
위 질의를 처리하고 난 후 0번 슬롯의 바이너리 상태는 아래와 같이 됩니다.



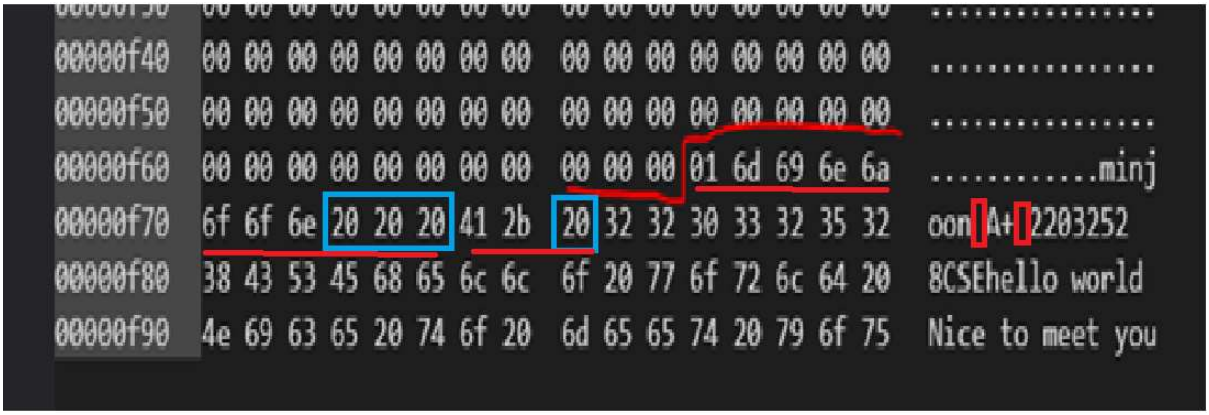
제일 앞의 00번 바이트에는 레코드(row)의 개수인 1이 들어가고,



그 후 01번, 02번 바이트에는 3947이라는 값이 들어가 있어서 0번째 레코드의 시작 주소는 3947바이트라는 것을 알 수 있습니다. 6b 0f가 little endian 방식으로 저장되기 때문에 0f6b를 10진수로 변환하면 3947입니다.



실제로 슬롯의 끝 부분에 가보면 3947바이트부터 레코드가 시작되는 것을 볼 수 있습니다.



레코드를 자세히 살펴 보면, name 컬럼은 CHAR(10)이라서, minjoon은 7바이트라서 3바이트 만큼 스페이스바로 채워 지는 것을 볼 수 있습니다.(파란색 네모 20 20 20),

그리고 마찬가지로 grade 컬럼도 CHAR(3)이기 때문에 2바이트인 A+ 이후에 스페이스바(공백)이 한 칸 있는 걸 확인할 수 있습니다.

그리고 그 이후의 컬럼들은 모두 VARCHAR이기 때문에 공백 없이 공간을 모두 활용하고 있는 걸 볼 수 있습니다.

사용자 질의 처리(Query Processing)

사용자가 입력할 수 있는 쿼리는 크게 3가지로 나눌 수 있습니다.

1. CREATE TABLE

CREATE TABLE은 다음과 같이 합니다.

CREATE TABLE tableName(student name CHAR(50) grade CHAR(3) sentence VARCHAR(100)

사용자의 테이블 생성 질의에서 tableName을 추출한 뒤 뒤의 컬럼들을 순차적으로 추출합니다.

그 후 테이블명으로 meta.meta파일에 해당하는 테이블이 어디에 기록될 지 추가해줍니다.

Metafile [2022-05-06 00:00:01] 생성일자 기록

student.meta ./student/ (메타 파일명, 폴더 경로)
player.meta ./player/
university.meta ./university/

그리고 student.meta파일을 작성합니다.

student.meta [2022-05-06 00:00:01] (생성일자 기록)

id CHAR(10)
name VARCHAR(20)
grade CHAR(3)

1. INSERT INTO

INSERT into student values("minjoon","A+","hello world this is minjoon","SW")

Insert 쿼리를 입력 받았을 시, 테이블명(student)를 우선 추출하고, 메타 파일을 이용해 해당 테이블의 정보를 가져옵니다. 레코드 삭제가 없다고 가정하면 가장 마지막 슬롯을 가져오고, 해당 슬롯의 헤더를 읽어서 빈 공간이 있으면 삽입하고 없으면 새로운 슬롯을 생성합니다. 그리고 슬롯의 헤더에 레코드의 시작 포인터를 2바이트로 가리키게 합니다. 그 후 메타 파일을 업데이트합니다.

레코드를 생성할 때 만약에 쿼리에 values()안에 null이 포함되어 있다면 몇 번째 컬럼인지 체크해서 그 컬럼에 해당하는 null bitmap의 비트를 1로 만들어줍니다.(null 체크)

2. SELECT

SELECT * FROM student WHERE name = "minjoon" AND grade = "A+"

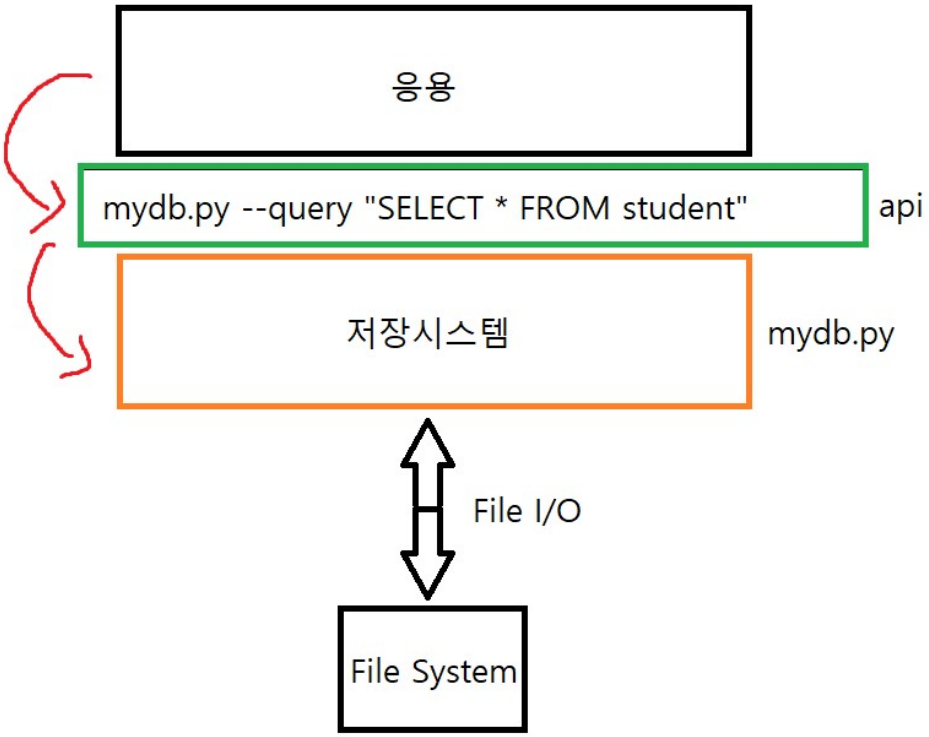
위와 같은 질의를 입력 받았을 경우 meta.meta파일을 읽어 student.meta 파일이 어느 경로에 있는 지 읽어옵니다.

그 후 student.meta파일을 읽어서 student테이블에 관련된 레코드가 있는 모든 슬롯 페이지의 리스트를 획득합니다.

우리의 DBMS에는 인덱스가 구현되어 있지 않기 때문에, 모든 슬롯을 반드시 확인해야 됩니다.

그 후 슬롯 단위로 하나씩 버퍼에 읽어들인 후 WHERE 절에 해당하는 조건의 레코드 인지 모든 레코드를 체크해서, 해당하는 레코드를 버퍼에 남겨놓습니다.

하나의 슬롯을 모두 읽었으면 다음 슬롯으로 이동해서 같은 작업을 반복해서, WHERE 절에 해당하는 레코드를 리턴합니다.



[사진] mydb.py 생성 후 질의 실행 후 파일 I/O를 나타낸 그림

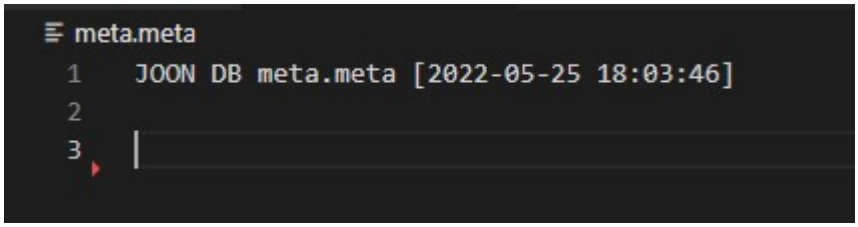
전체적인 그림으로 보면 위와 같은 시스템이 구축이 됩니다.

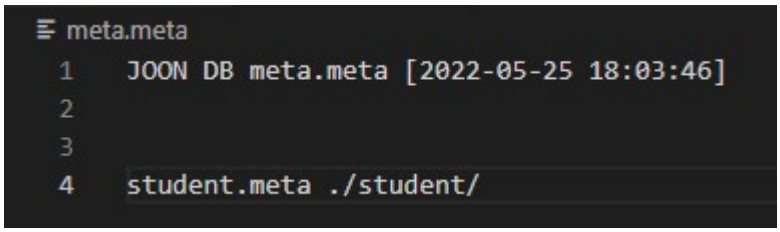
결과 – 기능 동작 정확성 검증 결과

1. 테이블 생성 쿼리 (CREATE TABLE)

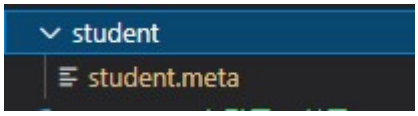
CREATE TABLE student(name CHAR(10) grade CHAR(3) department VARCHAR(20) introduction VARCHAR(50))

위 쿼리 실행 후,

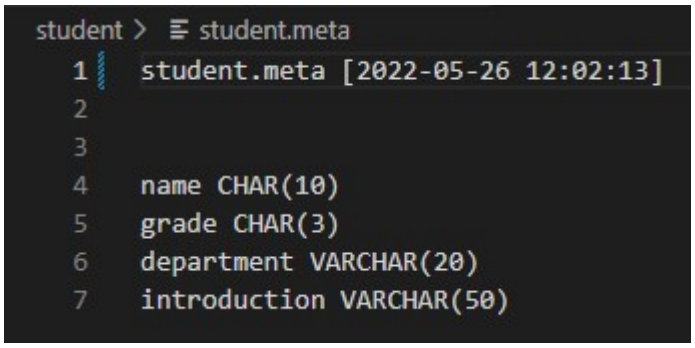




meta.meta파일에 student.meta ./student/ 메타 파일의 경로가 기록이 되었으며,



student.meta파일이 새로 생성되었으며,



student.meta파일에는 위와 같이 기록되었습니다.

그 후

CREATE TABLE subject(name VARCHAR(30) semester VARCHAR(10) department VARCHAR(20) introduction VARCHAR(50))

또 다른 테이블 subject 생성 쿼리를 실행하였더니,

```
1   JOON DB meta.meta [2022-05-25 18:03:46]
2
3
4   student.meta ./student/
5   subject.meta ./subject/
```

meta.meta 파일은 위와 같이 subject.meta가 추가되었고,

subject > subject.meta

```
1   subject.meta [2022-05-26 12:07:32]
2
3
4   name VARCHAR(30)
5   semester VARCHAR(10)
6   department VARCHAR(20)
7   introduction VARCHAR(50)
```

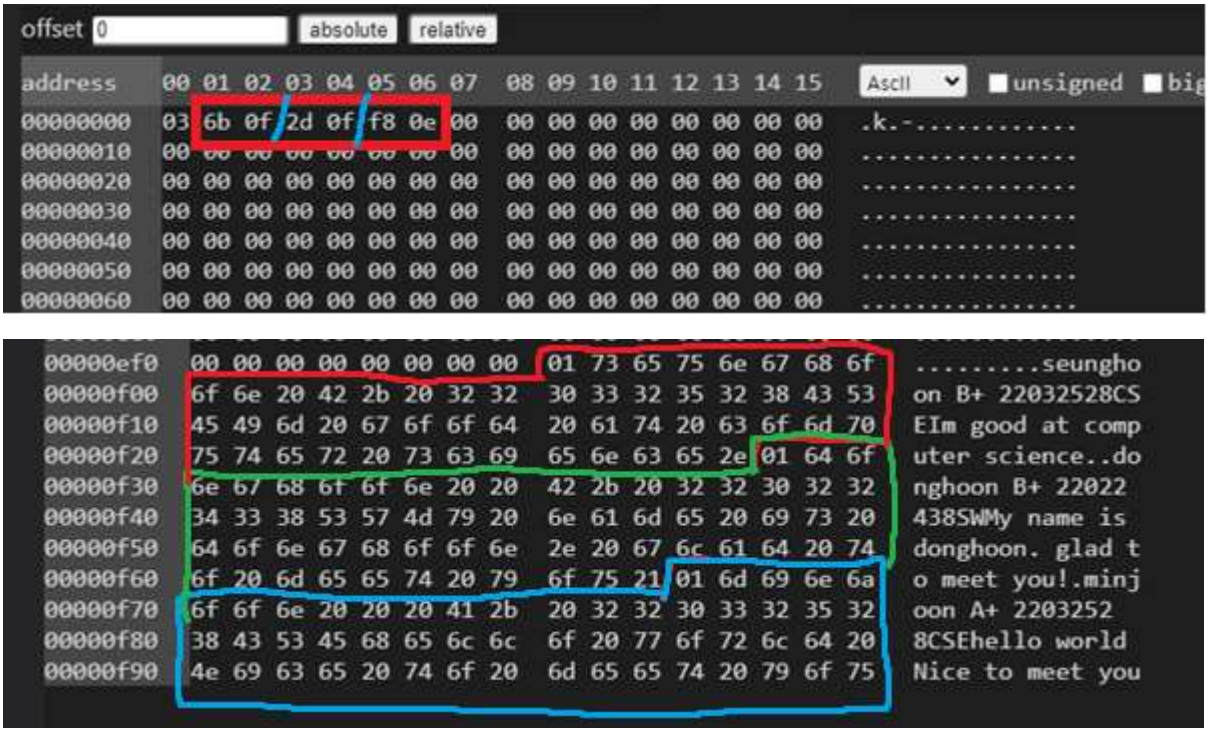
subject.meta파일은 위와 같이 생성되었습니다.

2. 레코드 삽입 쿼리 (INSERT INTO)

INSERT INTO student VALUES("minjoon","A+","CSE","hello world Nice to meet you")

INSERT INTO student VALUES("donghoon","B+","SW","My name is donghoon. glad to meet you!")

INSERT INTO student VALUES("seunghoon","B+","CSE","Im good at computer science.")



위의 INSERT 쿼리 3줄을 각각 실행하고 나면 student의 0번 슬롯에 위와 같이 슬롯의 시작 부분에 레코드 포인터 3개(각각 2바이트, 총 6바이트) , 그리고 슬롯의 끝 부분에 레코드 3개가 끝에 들어간 것을 확인할 수 있습니다.

'INSERT INTO student VALUES("donghoon","B+","SW","My name is donghoon. glad to meet you!")'

해당 쿼리를 대략 67회 이상 반복하면, 0번 슬롯이 레코드로 가득 차게 됩니다.

```
address   00 01 02 03 04 05 06 07  08 09 10 11 12 13 14 15   Ascii  unsigned  bigendian
00000000  3e 6b 0f 2d 0f f8 0e ba  0e 7c 0e 3e 0e 00 0e c2   >k.-.....|.>....
00000010  0d 84 0d 46 0d 08 0d ca  0c 8c 0c 4e 0c 10 0c d2   ...F.......N....
00000020  0b 94 0b 56 0b 18 0b da  0a 9c 0a 5e 0a 20 0a e2   ...V.↑.....^. ..
00000030  09 a4 09 66 09 28 09 ea  08 ac 08 6e 08 30 08 f2   ...f.(.....n.0..
00000040  07 b4 07 76 07 38 07 fa  06 bc 06 7e 06 40 06 02   ...v.8.....~.@..
00000050  06 c4 05 86 05 48 05 0a  05 cc 04 8e 04 50 04 12   .....H.......P..
00000060  04 d4 03 96 03 58 03 1a  03 dc 02 9e 02 60 02 22   .....X.→.....`."
00000070  02 e4 01 a6 01 68 01 2a  01 ec 00 ae 00 00 00 00   .....h.*........
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 64   ...............d
000000b0  6f 6e 67 68 6f 6f 6e 20  20 42 2b 20 32 32 30 32   onghoon B+ 2202
000000c0  32 34 33 38 53 57 4d 79  20 6e 61 6d 65 20 69 73   2438SWMy name is
000000d0  20 64 6f 6e 67 68 6f 6f  6e 2e 20 67 6c 61 64 20   donghoon. glad
000000e0  74 6f 20 6d 65 65 74 20  79 6f 75 21 01 64 6f 6e   to meet you!.don
000000f0  67 68 6f 6f 6e 20 20 42  2b 20 32 32 30 32 32 34   ghoon B+ 220224
00000100  33 38 53 57 4d 79 20 6e  61 6d 65 20 69 73 20 64   38SWMy name is d
00000110  6f 6e 67 68 6f 6f 6e 2e  20 67 6c 61 64 20 74 6f   onghoon. glad to
00000120  20 6d 65 65 74 20 79 6f  75 21 01 64 6f 6e 67 68   meet you!.dongh
00000130  6f 6f 6e 20 20 42 2b 20  32 32 30 32 32 34 33 38   oon B+ 22022438
00000140  53 57 4d 79 20 6e 61 6d  65 20 69 73 20 64 6f 6e   SWMy name is don
00000150  67 68 6f 6f 6e 2e 20 67  6c 61 64 20 74 6f 20 6d   ghoon. glad to m
00000160  65 65 74 20 79 6f 75 21  01 64 6f 6e 67 68 6f 6f   eet you!.donghoo
00000170  6e 20 20 42 2b 20 32 32  30 32 32 34 33 38 53 57   n B+ 22022438SW
00000180  4d 79 20 6e 61 6d 65 20  69 73 20 64 6f 6e 67 68   My name is donghh
00000190  6f 6f 6e 2e 20 67 6c 61  64 20 74 6f 20 6d 65 65   oon. glad to mee
000001a0  74 20 79 6f 75 21 01 64  6f 6e 67 68 6f 6f 6e 20   t you!.donghoon
000001b0  20 42 2b 20 32 32 30 32  32 34 33 38 53 57 4d 79   B+ 22022438SWMy
000001c0  20 6e 61 6d 65 20 69 73  20 64 6f 6e 67 68 6f 6f   name is donghoo
000001d0  6e 2e 20 67 6c 61 64 20  74 6f 20 6d 65 65 74 20   n. glad to meet
```

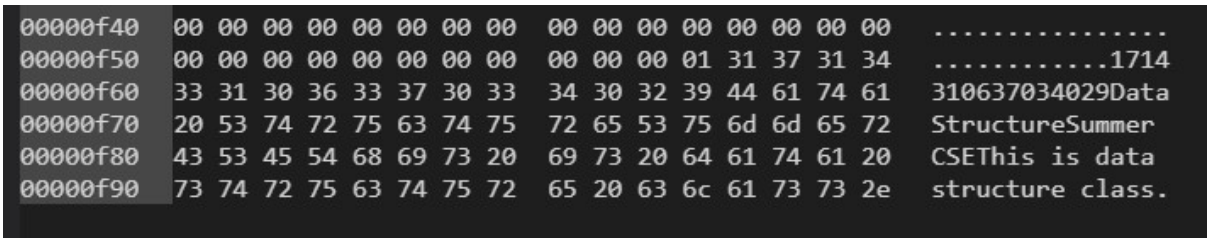
해당 INSERT 질의를 실행하면 새로 생성되는 레코드의 크기는

CHAR(10)+CHAR(3)+VARCHAR+VARCHAR = 10+3+2+38 = 53바이트로,

슬롯에 빈 공간이 53바이트 + 2바이트(레코드 포인터) = 55바이트의 공간이 필요한데 위 사진에는 50바이트밖에 빈 공간이 없습니다. 따라서 1번 슬롯을 새로 생성해서 그 슬롯에 레코드를 삽입하게 됩니다.

그리고 subject 테이블에 아래의 INSERT 질의를 실행하게 되면,

INSERT INTO subject VALUES("Data Structure","Summer","CSE","This is data structure class.")



```
00000f40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000f50  00 00 00 00 00 00 00 00  00 00 00 01 31 37 31 34   ............1714
00000f60  33 31 30 36 33 37 30 33  34 30 32 39 44 61 74 61   310637034029Data
00000f70  20 53 74 72 75 63 74 75  72 65 53 75 6d 6d 65 72   StructureSummer
00000f80  43 53 45 54 68 69 73 20  69 73 20 64 61 74 61 20   CSEThis is data
00000f90  73 74 72 75 63 74 75 72  65 20 63 6c 61 73 73 2e   structure class.
```

위와 같이 슬롯 바이너리 파일에 삽입이 되는데, 1714로 데이터가 시작이 됩니다. 왜냐면 subject 테이블에는 CHAR 데이터 형이 없이 모든 컬럼이 VARCHAR이기 때문에 가변길이 레코드의 오프셋과 사이즈가 먼저 나오게 됩니다.

3. 레코드 검색 쿼리 (SELECT FROM)

위에서 생성한 레코드들을 검색하는 질의 테스트를 보여드리겠습니다.

SELECT * FROM student

```
67 Rows Selected.
                name                    grade                   department                  introduction
----------------------------------------------------------------------------------------------------------------
             minjoon                      A+                          CSE          hello world Nice to meet you
             donghoon                     B+                           SW        My name is donghoon. glad to m
             seunghoon                    B+                          CSE          Im good at computer science.
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
             donghoon                     B+                           SW   My name is donghoon. glad to meet you!
```

SELECT * FROM student WHERE name = "minjoon"

```
PS C:\Users\song\dbproject> python main.py 'SELECT * FROM student WHERE name = "minjoon"'
1 Rows Selected.
                name                    grade                   department                  introduction
----------------------------------------------------------------------------------------------------------------
             minjoon                      A+                          CSE          hello world Nice to meet you
PS C:\Users\song\dbproject> ^ ^
```

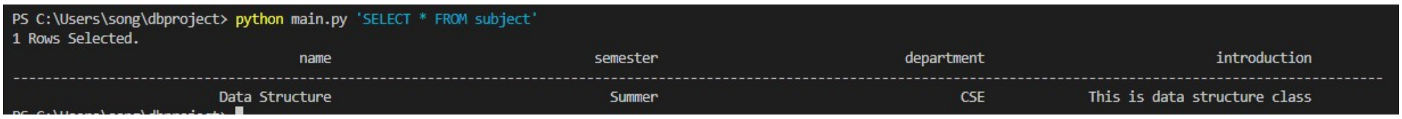
SELECT * FROM student WHERE name = "seunghoon" AND grade = "B+"

```
PS C:\Users\song\dbproject> python main.py 'SELECT * FROM student WHERE name = "seunghoon" AND grade = "B+"'
1 Rows Selected.
                name                    grade                   department                  introduction
----------------------------------------------------------------------------------------------------------------
             seunghoon                    B+                          CSE          Im good at computer science.
```

SELECT name grade FROM student WHERE name = "seunghoon" AND grade = "B+"

```
PS C:\Users\song\dbproject> python main.py 'SELECT name grade FROM student WHERE name = "seunghoon" AND grade = "B+"'
1 Rows Selected.
                name                    grade
------------------------------------------------------
             seunghoon                    B+
```

SELECT semester FROM subject



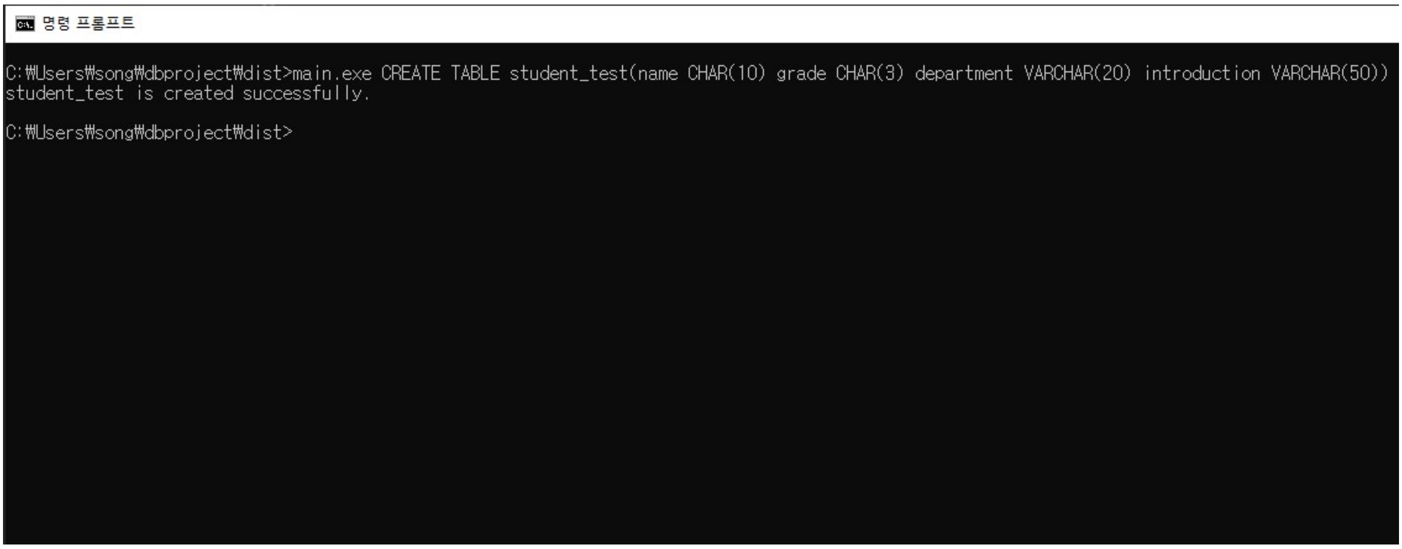
SELECT * FROM subject

구현 언어 및 개발 환경

Python 3.10.x

Visual Studio Code

실행 방법



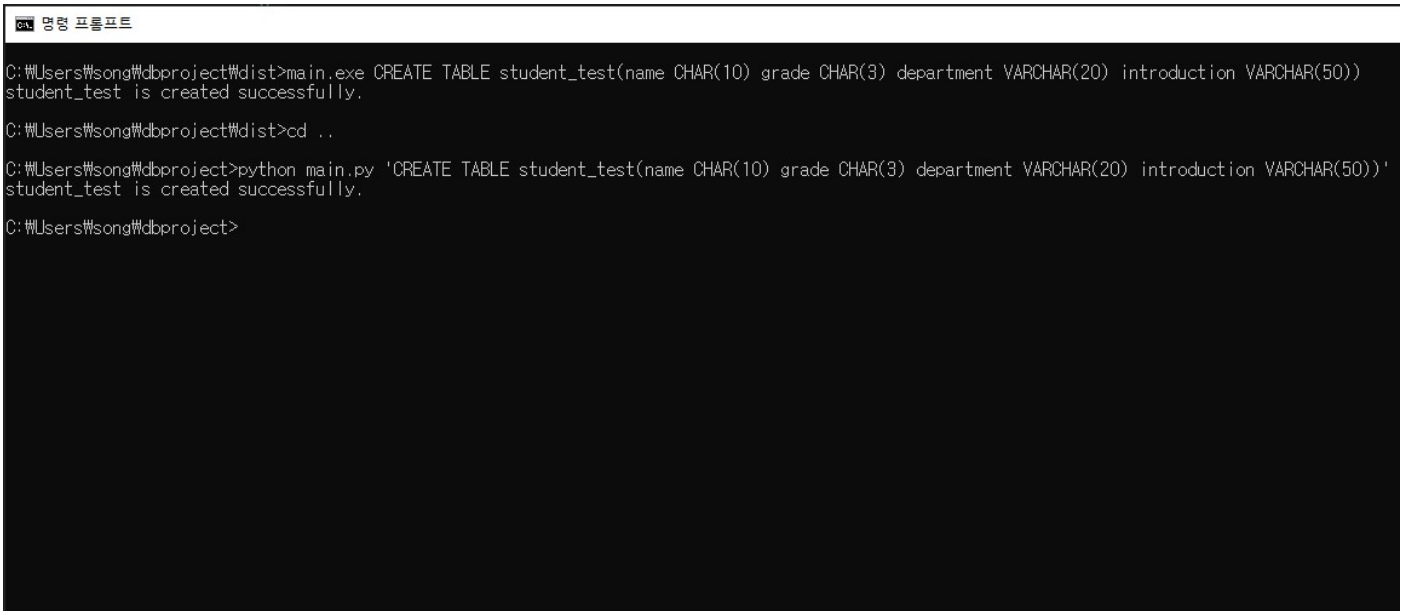
윈도우 기준

명령 프롬프트 : main.exe파일이 있는 경로에 가서

main.exe 쿼리 내용

ex)

main.exe CREATE TABLE student_test(name CHAR(10) grade CHAR(3) department VARCHAR(20) introduction VARCHAR(50))



혹은 main.py 파이썬 파일로 실행하는 경우

python main.py '쿼리 내용'

ex)

python main.py 'CREATE TABLE student_test(name CHAR(10) grade CHAR(3) department VARCHAR(20) introduction VARCHAR(50))'

소스 코드 (main.py)

```
# -*- coding: utf-8 -*-

import binascii
import struct
import sys
from datetime import datetime
import os
from os.path import exists
import re

from bitarray import bitarray

sys.path.append(os.path.dirname(os.path.abspath(os.path.dirname(__file__))))

SLOT_SIZE = 4000

def err(errstr):
    print("Exception ocurred : ", errstr)
    sys.exit(0)

def lpad(i, width, fillchar='0'):
    """입력된 숫자 또는 문자열 왼쪽에 fillchar 문자로 패딩"""
    return str(i).rjust(width, fillchar)

def rpad(i, width, fillchar='0'):
    """입력된 숫자 또는 문자열 오른쪽에 fillchar 문자로 패딩"""
    return str(i).ljust(width, fillchar)

def stringToBinary(string):
    return ' '.join(format(ord(c), 'b') for c in string)

def processQuery(query):
    if "CREATE TABLE " in query.upper():
        tableName = query.split("CREATE TABLE ")[1].split("(")[0].strip()
        args = []
        argsArr = query[query.find("(")+1:].split(") ")

        for arg in argsArr:
            try:
                argName = arg.split(" ")[0]
                argPref = arg.split(" ")[1]
                if argPref[-1] != ")":
                    argPref = argPref + ")"
                if argPref[-1] == ")" and argPref[-2] == ")":
                    argPref = argPref[:-1]

                args.append((argName,argPref))
            except:
                print('',end='')
```

```
        print( createTable(tableName, args) )

    elif "INSERT INTO" in query.upper():
        tableName = query.split("INSERT INTO ")[1].split("VALUES")[0].strip()
        args =
query.split("VALUES")[1].strip().split("(")[1].split(")")[0].split(",")
        newArgs = []
        for arg in args:
            arg = arg.strip('"')
            newArgs.append(arg)

        print( insertRow(tableName, args) )

    elif "SELECT " in query.upper():
        tableName = query.split("FROM ")[1].split("WHERE")[0].strip()
        selectCols = query.split("SELECT")[1].split("FROM")[0].strip().split("
")
        try:
            conds = query.split("WHERE")[1].strip().split("AND")
        except:
            conds = []

        condsArr = []

        for cond in conds:
            colName = cond.split("=")[0].strip()
            value = cond.split("=")[1].strip().strip('"')
            condsArr.append((colName,value))

        result = ( selectTable(tableName,selectCols, selectCols[0]=="*",
condsArr) )

        print( len(result),"Rows Selected." )

        if len(result) <= 0:
            return

        cols = result[0].keys()
        for col in cols:
            print("%40s " %(col), end='')
        print("")

        for col in cols:
            print("----------------------------------------",end="")
        print("")

        for res in result:
            for col in cols:
                print("%40s " %(res[col]), end='')
            print("")
```

```
def createDirectory(directory):
    try:
        if not os.path.exists(directory):
            os.makedirs(directory)
    except OSError:
        err("Error: Failed to create the directory.")


def createTable(tableName, args):

    # 메타메타 파일 열자

    try:
        meta = open("./meta.meta", 'r+')
    except:
        meta = open("./meta.meta", "w")
        meta.writelines("JOON DB meta.meta [" +
datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "]")
        meta.writelines("\n")
        meta.writelines("\n")
        meta.flush()
        meta.close()
        meta = open("./meta.meta", 'r+')

    while True:
        readData = meta.readline()

        if readData != "":
            currTableName = readData.split(".meta")[0]
            if tableName == currTableName:
                err("tableName already exists")
        else:
            break

    meta.writelines("\n" + tableName + ".meta ./" + tableName + "/")
    meta.flush()
    meta.close()

    # 테이블 폴더 만들자
    createDirectory("./" + tableName)
    # 메타 만들자

    meta = open("./" + tableName + "/" + tableName + ".meta", "a+")
    meta.writelines(tableName + ".meta [" +
datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "]")
    meta.writelines("\n")
    meta.writelines("\n")

    for arg in args:
        meta.writelines("\n" + arg[0] + " " + arg[1])

    return tableName+" is created successfully."
```

```
def insertRow(tableName, args):

    # 메타에 쓰이는 3종 세트
    cols = []
    types = []
    sizes = []
    colData = []
    # 우선 테이블 메타 파일 열어야됨

    try:
        reader = open("./" + tableName + "/" + tableName + ".meta", "r")
    except:
        err(tableName + " meta not exists")

    while True:
        readData = reader.readline()

        if " CHAR(" in readData or " VARCHAR(" in readData:
            cols.append(readData.split(' ')[0])
            types.append(readData.split(' ')[1].split('(')[0])
            sizes.append(int(readData.split('
')[1].split('(')[1].split(')')[0]))

            temp = {}
            temp["cols"] = readData.split(' ')[0]
            temp["types"] = readData.split(' ')[1].split('(')[0]
            temp["sizes"] = int(readData.split('
')[1].split('(')[1].split(')')[0])

            colData.append(temp)
        if readData == "":
            break

    nullBitMap = ""
    nullBitMapArr = []
    record = ""
    varcharOffset = 1  #맨 앞에 null bitmap 이 있기 때문에

    varcharOffset += len(list(filter(lambda x: x == "VARCHAR", types))) * 4
    varcharOffset += sum( [x["sizes"] for x in colData if x["types"] ==
"CHAR"] )

    for i in range(len(args)):
        #empty string or null is preprocessed
        #convert to variable length record
        arg = args[i]

        if len(arg) > sizes[i]:
            err("column "+cols[i]+" exceed length limit
"+str(sizes[i])+"  ("+str(len(arg))+")")

        if arg is None or arg == "NULL":
            nullBitMap += "1"
```

```
            arg = ''
            args[i] = ''
        else:
            nullBitMap += "0"

        if types[i] == "CHAR":
            record += rpad(arg,sizes[i],' ')
        elif types[i] == "VARCHAR":
            record += str(varcharOffset)+lpad(str(len(arg)),2,'0')
            varcharOffset += len(arg)

    for i in range(len(args)):
        arg = args[i]
        if types[i] == "VARCHAR":
            record += arg

    # args = ["minjoon", "A+", "hello world this is mslacinjoon", "SW"]
    for i in range(len(nullBitMap)):
        if nullBitMap[i] == "0":
            nullBitMapArr.append(0)
        else:
            nullBitMapArr.append(1)


    nullBitMap = rpad(nullBitMap,8,'0')


    nullDecimal = chr(int(nullBitMap,2)+1)


    record = nullDecimal + record

    #이제 슬롯을 찾아서 레코드를 넣을거임

    slotCurr = 0
    while True:
        if insertSlot(tableName, slotCurr, record) == 1: #성공했을 시 루프
빠져나간다..
            break
        slotCurr += 1

    return tableName+ " inserted 1 row successfully."

def checkSlot(tableName, slotNum):
    return exists("./" + tableName + "/slot" + lpad(slotNum,3,'0') + ".bin")

def createSlot(tableName, slotNum):
    slot = open("./" + tableName + "/slot" + lpad(slotNum,3,'0') + ".bin",
"wb+")

    slot.seek(0)
    slot.write(lpad("\0",4000,'\0').encode())

    slot.seek(0)
```

```
    slot.write(struct.pack('b',0))

    slot.flush()
    slot.close()


def selectTable(tableName, columns, selectAll=True, conditions=[]):
    returnData = []

    # 메타에 쓰이는 3종 세트
    cols = []
    types = []
    sizes = []
    colData = []
    # 우선 테이블 메타 파일 열어야됨

    try:
        reader = open("./" + tableName + "/" + tableName + ".meta", "r")
    except:
        err(tableName + " meta not exists")

    while True:
        readData = reader.readline()

        if " CHAR(" in readData or " VARCHAR(" in readData:
            cols.append(readData.split(' ')[0])
            types.append(readData.split(' ')[1].split('(')[0])
            sizes.append(int(readData.split('
')[1].split('(')[1].split(')')[0]))

            temp = {}
            temp["cols"] = readData.split(' ')[0]
            temp["types"] = readData.split(' ')[1].split('(')[0]
            temp["sizes"] = int(readData.split('
')[1].split('(')[1].split(')')[0])

            colData.append(temp)
        if readData == "":
            break

    slotNum = 0

    while True:
        try:
            slot = open("./" + tableName + "/slot" + lpad(slotNum,3,'0') +
".bin", "rb")
        except:
            break

        recordPointer = -1
        i = 1
        while i < SLOT_SIZE-1:
            tempReturnCol = {}
```

```
            slot.seek(i)
            readData = slot.read(2)
            currRecordPointer = int(struct.unpack("H",readData)[0])+1
            nextRecordPointer = -1

            if i < SLOT_SIZE-3:
                slot.seek(i+2)
                readData2 = slot.read(2)
                nextRecordPointer = int(struct.unpack("H",readData2)[0])

            if nextRecordPointer == 0:
                nextRecordPointer = SLOT_SIZE

            if currRecordPointer <= 1:
                break

            slot.seek(currRecordPointer-1)
            buf = slot.read(abs(nextRecordPointer-currRecordPointer))
            temp = abs(nextRecordPointer-currRecordPointer)
            columnData = str(buf.decode())
            nullBitMap = lpad(str(bin(buf[0]-1)).split('0b')[1],8,'0')
            columnIter = 1

            for idx, type in enumerate(types):
                if cols[idx] not in columns and selectAll == False:
                    if type == "CHAR":
                        columnIter += sizes[idx]
                    elif type == "VARCHAR":
                        columnIter += 4
                    #생략해도 되는 컬럼.. 값 리턴 안해줘도 된다.
                    continue
                else:
                    #리턴 데이터에 넣어 줘야되는 컬럼
                    if type == "CHAR":
                        if nullBitMap[idx] == "1":
                            tempReturnCol[cols[idx]]="NULL"
                        else:
                            tempReturnCol[cols[idx]]=columnData[columnIter:colum
nIter+sizes[idx]]
                            columnIter += sizes[idx]
                    elif type == "VARCHAR":
                        if nullBitMap[idx] == "1":
                            tempReturnCol[cols[idx]]="NULL"
                            columnIter += 4
                        else:
                            startIdx = int(columnData[columnIter:columnIter+2])
                            size = int(columnData[columnIter+2:columnIter+4])
                            tempReturnCol[cols[idx]]=columnData[startIdx:startId
x+size]
                            columnIter += 4

            flag = False
```

```
            for cond in conditions:
                # print(str(cond)+"@"+tempReturnCol[cond[0]]+"@"+cond[1]+"@" )
                if tempReturnCol[cond[0]].rstrip() != cond[1]:
                    #조건 하나라도 안맞으면 안넣고 continue
                    flag = True
                    break

            if flag == False:
                returnData.append(tempReturnCol)
            i += 2
        slotNum += 1

    return returnData


#특정 슬롯에 레코드를 삽입하는 함수, 공간이 부족해서 실패하면 -1 리턴
def insertSlot(tableName, slotNum, record):
    if checkSlot(tableName, slotNum) == False:
        #없으면 만들자
        createSlot(tableName, slotNum)

    try:
        slot = open("./" + tableName + "/slot" + lpad(slotNum,3,'0') + ".bin",
"rb+")
    except:
        err("table "+ tableName+ " slot "+slotNum +" not exists ")

    #빈공간 찾아야됨

    slot.seek(0)
    count = slot.read(1)
    count = int(struct.unpack('b',count)[0])


    emptyPointer = -1
    slot.seek(SLOT_SIZE) ##마지막으로 커서를 옮겨서 한 바이트씩 읽어서 \0 이
아닐때까지 읽는다

    for i in range(0,SLOT_SIZE-len(record)-2-1-count*2): #레코드 포인터가
2 바이트라서 2 를 추가적으로 뺐다( 레코드 포인터의 여유 공간까지 고려 )
        slot.seek(SLOT_SIZE-i)

        readData = slot.read(1)

        if readData == b'\x00':
            emptyPointer = SLOT_SIZE-i
            if i != 0:
                emptyPointer += 1
            break

    if emptyPointer == -1:
        slot.close()
```

```
        return -1  # 해당 슬롯에 빈공간이 없어서 -1 리턴

    startPoint = emptyPointer-len(record)
    slot.seek(startPoint)
    slot.write(record.encode())

    slot.seek(0)

    recordPointer = -1
    i = 1
    while i < SLOT_SIZE-1:
        slot.seek(i)
        readData = slot.read(2)
        data = int(struct.unpack("H",readData)[0])

        if data == 0:
            recordPointer = i
            break
        i += 2


    if recordPointer == -1:
        err("슬롯 공간은 있는데 레코드 포인터가 없는 말도 안되는 경우")
    slot.seek(recordPointer)
    slot.write( struct.pack('H',startPoint)  )


    #삽입에 성공했으므로 슬롯 헤더에 레코드 갯수를 1 올려줘야 됨 .
    slot.seek(0)
    count = slot.read(1)
    count = int(struct.unpack('b',count)[0])

    slot.seek(0)
    slot.write(struct.pack('b',count+1))

    slot.close()

    return 1

if __name__ != "__main__":
    sys.exit(0)

queryString = ""
for idx, arg in enumerate(sys.argv):
    if idx == 0:
        continue
    queryString += arg+" "

processQuery(queryString)
```