CmpE 321 Introduction to Database Systems
2018 Spring
Assignment 2
Implementing Storage Manager System

Baran Kılıç
2014400123

April 9, 2018

1 Introduction

In this assignment, we are expected to implement a simple storage manager. I used C++ as programming language and compiled and tested my code on Ubuntu. We have 10 MB storage area to use. We assume that user always enter valid data. Supported operations are creating, deleting, listing types and creating, deleting, searching and listing records.

I created classes for type, system catalog, record, page and file. I created binary files to store the information in classes. I read record pages page by page.

When I create types and records, I add them to the first empty space. When I delete them, I only set the full bit to false. I find the empty space by linear search. When I search a record, I also use linear search.

2 Changes From The Initial Design

I made some changes to decrease system catalog size. These are the updated constraints:

- Max length of a type name is 12 characters.
- Max length of a field name is 12 characters.
- Max number of fields for a type is 8.
- Max file size is 64 kilobytes. (64 pages).

Since the file header is removed from the first assignment. I updated the following constraint:

- System catalog header contains the number of types, the index of the next empty space for a type, an array that stores type ids for each file and the index of the next empty file.

I removed file id from type header since there can be multiple files. This information is now stored in system catalog. I added field number for a type in type header since the user may want to create a record with less fields than maximum allowed fields. Constraints that I forgot to write in the first assignment:

- Minimum number of primary key for a type is 1.
- Maximum number of primary key for a type is 8.
- Minimum number of field for a type is 1.

2.1 Data Structures (updated)

2.1.1 System Catalog

Type count	Next empty index	Next empty file index	Type id of file 1	...	Type id of file 159

Full empty info	Type name	# of fields	Primary key bitmap	Field name 1	...	Field name 8
Full empty info	Type name	# of fields	Primary key bitmap	Field name 1	...	Field name 8
⋮						

2.1.2 File

Page 1
Page 2
⋮
Page 64

2.1.3 Page

Page Header			**Remaining page**		
Is next page used	# of records	Next empty record index	Record 1	...	Record 28

2.1.4 Record

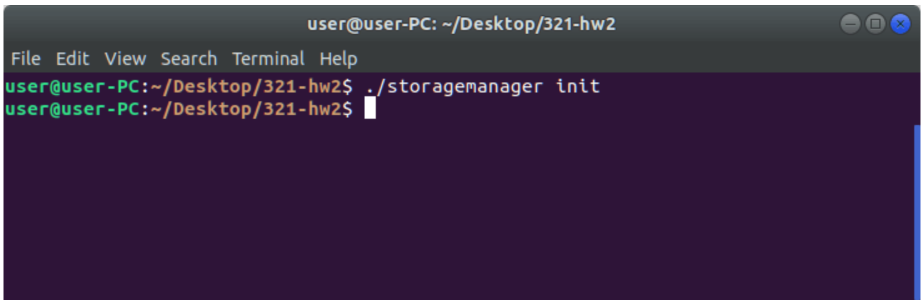
Record Header	**Remaining record**		
Full/empty info	Field 1	...	Field 8

3 Sample Usage and Outputs

```
./storagemanager help
```

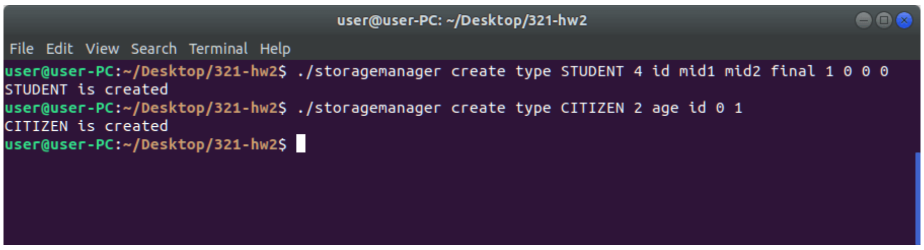
can be run to see usage examples and command syntax

3.1 Initialization


user@user-PC: ~/Desktop/321-hw2
File Edit View Search Terminal Help

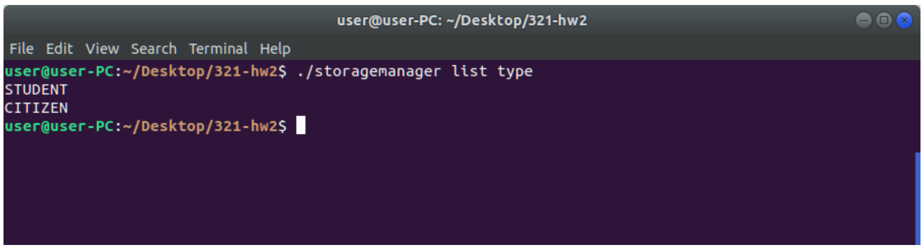
```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager init
user@user-PC:~/Desktop/321-hw2$ 
```

3.2 Create Type


user@user-PC: ~/Desktop/321-hw2
File Edit View Search Terminal Help

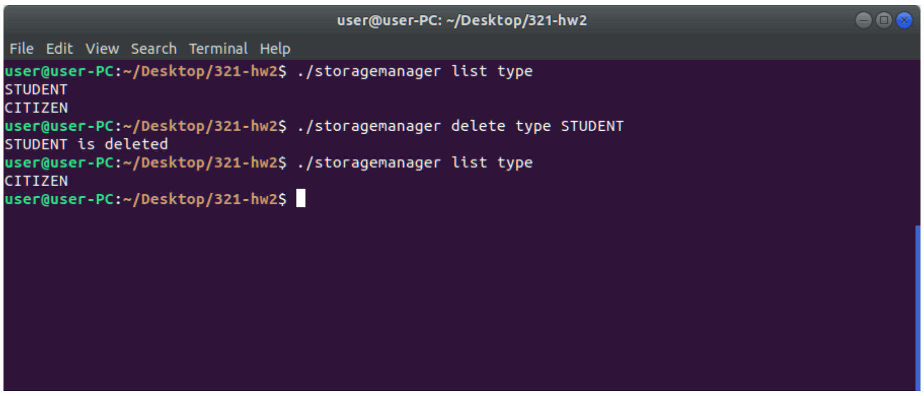
```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create type STUDENT 4 id mid1 mid2 final 1 0 0 0
STUDENT is created
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create type CITIZEN 2 age id 0 1
CITIZEN is created
user@user-PC:~/Desktop/321-hw2$ 
```

3.3 List Type


user@user-PC: ~/Desktop/321-hw2
File Edit View Search Terminal Help

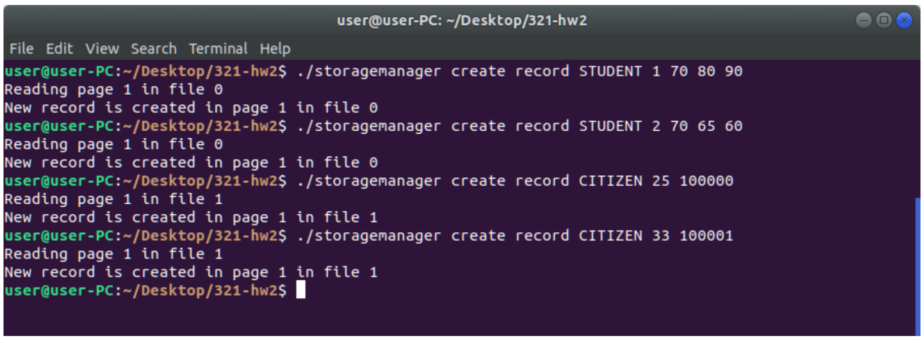
```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list type
STUDENT
CITIZEN
user@user-PC:~/Desktop/321-hw2$ 
```

3.4 Delete Type


user@user-PC: ~/Desktop/321-hw2
File Edit View Search Terminal Help

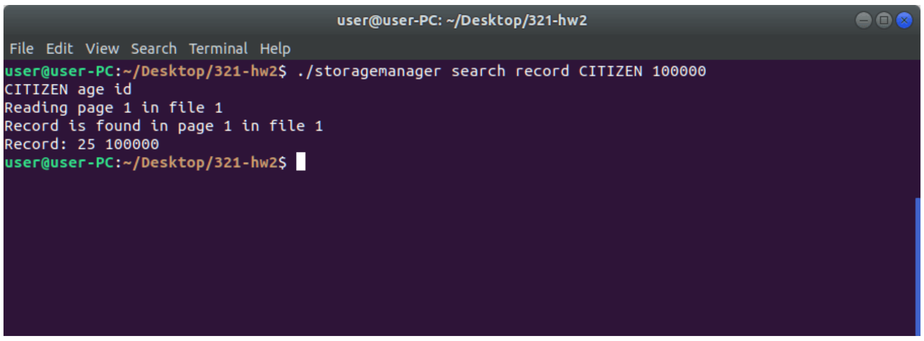
```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list type
STUDENT
CITIZEN
user@user-PC:~/Desktop/321-hw2$ ./storagemanager delete type STUDENT
STUDENT is deleted
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list type
CITIZEN
user@user-PC:~/Desktop/321-hw2$ 
```

3.5 Create Record


user@user-PC: ~/Desktop/321-hw2
File Edit View Search Terminal Help

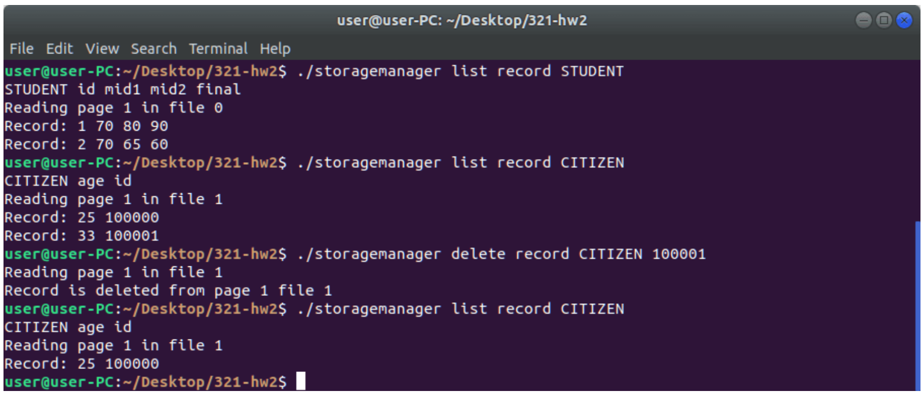
```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create record STUDENT 1 70 80 90
Reading page 1 in file 0
New record is created in page 1 in file 0
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create record STUDENT 2 70 65 60
Reading page 1 in file 0
New record is created in page 1 in file 0
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create record CITIZEN 25 100000
Reading page 1 in file 1
New record is created in page 1 in file 1
user@user-PC:~/Desktop/321-hw2$ ./storagemanager create record CITIZEN 33 100001
Reading page 1 in file 1
New record is created in page 1 in file 1
user@user-PC:~/Desktop/321-hw2$ 
```

3.6 Search Record



```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager search record CITIZEN 100000
CITIZEN age id
Reading page 1 in file 1
Record is found in page 1 in file 1
Record: 25 100000
user@user-PC:~/Desktop/321-hw2$
```

3.7 List and Delete Record



```
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list record STUDENT
STUDENT id mid1 mid2 final
Reading page 1 in file 0
Record: 1 70 80 90
Record: 2 70 65 60
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list record CITIZEN
CITIZEN age id
Reading page 1 in file 1
Record: 25 100000
Record: 33 100001
user@user-PC:~/Desktop/321-hw2$ ./storagemanager delete record CITIZEN 100001
Reading page 1 in file 1
Record is deleted from page 1 file 1
user@user-PC:~/Desktop/321-hw2$ ./storagemanager list record CITIZEN
CITIZEN age id
Reading page 1 in file 1
Record: 25 100000
user@user-PC:~/Desktop/321-hw2$
```

4 Conclusion and Assessment

I created a very simple storage manager. It does the basic operations like adding, removing and listing types and records.

Since the description said "do not include extra storage structures", I didn't implemented indexing. Therefore, my storage manager is slow and searches sequentially for records. To simplify the implementation, I used fixed length fields and stored each type in a different file.

When I delete a record, I simply set its full bit (not a bit in reality, its is one byte since the smallest size is one byte for a data type in C++) to false. I do not move any record to fill this gap. This makes deletion fast. However, when I list or search records, I have to skip these deleted records. To do this, I need to check their full bit. This makes listing and searching slow. Another result of not moving records while deleting records is that file number for a type increases but not decreases. The only option for a file belonging to a type to be deleted is being fully empty. If a file do not contain any record, I release this file (I mean deletion) to be used by other types. This behaviour is not ideal. The files may end up having very few records and the records may be scattered all over the files. There should be a way to move them to partially empty files so that we utilized files better.