CMPE 321

ASSIGNMENT 2

STORAGE MANAGER SYSTEM IMPLEMENTATION

Cemal Burak Aygün

2014400072

2018 SPRING

Contents

1 INTRODUCTION

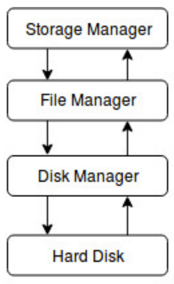
A **Storage Manager System** is a program used for storing data on disk in an organized structure. It provides its users with secure and convenient ways to access, modify and delete data.

In the previous assignment, I designed a simple storage manager system where users can create, access and delete data of several types specified by them. The system consists of a **System Catalog** which stores metadata about the system and data files that stores pages of records (data).

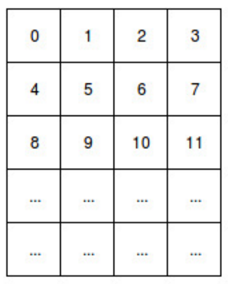
In this assignment, I implemented my design in **JAVA** as a **Terminal** application. In my implementation, there are 3 basic structures:

- **Disk Manager** is responsible for accessing the hard disk drive. It can retrieve a page from HDD once its address (ID) is provided. It does not know the internal of a page.
- **File Manager** is responsible for creating and deleting files and allocating pages to files. Once a page is allocated to a file, it remains allocated to that file (even if it becomes empty after some time) until the user deletes the file. It knows the internal of a page. It can process pages by inspecting page headers. It can retrieve/insert/delete records of/into/from a page but it does not know about the internal of a record.
- **Storage Manager** is responsible for interacting with user and providing user with common DDL and DML operations. It knows about the internal of a record.

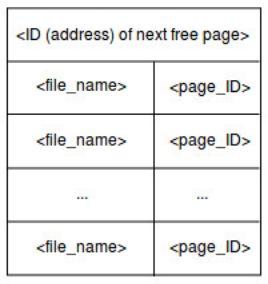
In my implementation, a single text file with name **DB_DISK** is created and maintained. This file represents the hard disk drive. File Manager reserves the first page (Page#0) to itself and stores a **Disk Directory** on that page. Disk Directory keeps the information of the **next free page ID** and **<file name, page ID> pairs**. All the other pages of the HDD are used for storing System Catalog and data files. *(See the figure below.)* Also, once the program starts, File Manager loads Disk Directory from Page#0 into its memory and every reading of Disk Directory is done from the memory throughout the program. After an update to Disk Directory in memory, it is saved to the disk (Page#0) immediately.



(a) Interaction Scheme



(b) Hard Disk Abstraction



(c) Structure of Disk Directory

Although I designed the data structures of my system in terms of **bytes**, I didn't work with real binary files in my implementation. I simulated the binary structures in a single text file in which I used several separator characters for separating pages in hard disk from each other, separating records in a page from each other and separating fields in a record from each other. Furthermore, as in the design, I assumed the user always provides valid inputs. Hence, I didn't implement any error checking.

2 CHANGES FROM THE INITIAL DESIGN

In the previous assignment, I designed System Catalog file as a single structure without any pages. I changed it to include page structures just as a Data File in order to keep the structures uniform in Storage Manager. In this assignment, structure of a file (either Data File or System Catalog file) is as follows:

<page structure>	<page structure>	...	<page structure>
...	...	...	...
<page structure>	<page structure>	...	<page structure>

Structure of a page is as follows:

pageID	ptrToNextPage	isEmpty	recNum
<record structure>	<record structure>	...	<record structure>
...	...	...	...
<record structure>	<record structure>	...	<record structure>

A **<record structure>** for a Data File is as follows :

field1 value (4)	...	fieldN value (4)

A **<record structure>** for System Catalog file is as follows:

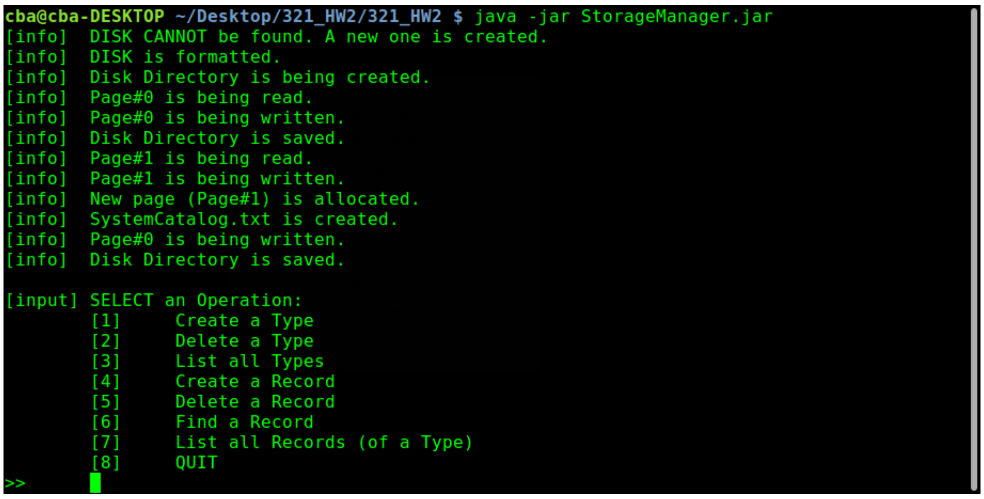
typeName	maxRecNumInPage	fieldNum	field1Name	...	field8Name

I didn't change my initial design except for the structure of System Catalog file.

3 SAMPLE USAGE AND OUTPUTS

Run the following command on a **Terminal**:

java -jar StorageManager.jar



```
cba@cba-DESKTOP ~/Desktop/321_HW2/321_HW2 $ java -jar StorageManager.jar
[info]  DISK CANNOT be found. A new one is created.
[info]  DISK is formatted.
[info]  Disk Directory is being created.
[info]  Page#0 is being read.
[info]  Page#0 is being written.
[info]  Disk Directory is saved.
[info]  Page#1 is being read.
[info]  Page#1 is being written.
[info]  New page (Page#1) is allocated.
[info]  SystemCatalog.txt is created.
[info]  Page#0 is being written.
[info]  Disk Directory is saved.

[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>
```

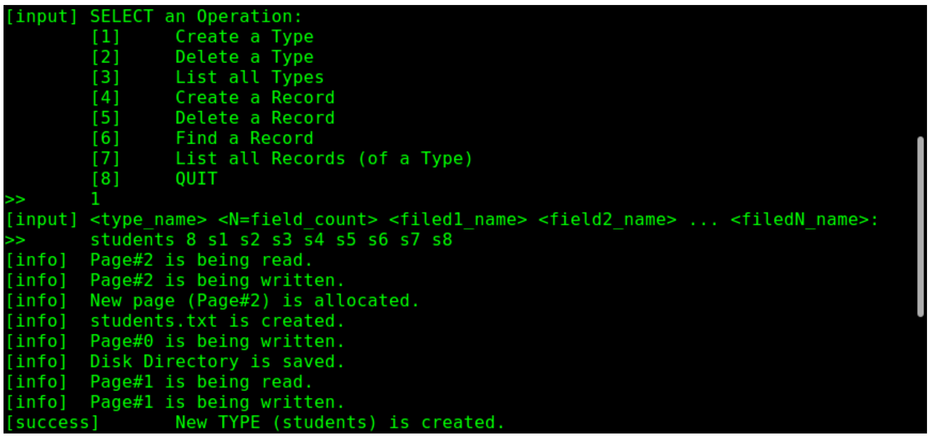
A text file with name **DB_DISK** will be created in the directory of **StorageManager.jar** which represents the hard disk drive. Then, **File Manager** creates a Disk Directory on Page#0 and **Storage Manager** creates

System Catalog file (to which Page#1 is allocated). Then, main menu is displayed for user to select an operation.

3.1 Creating a Type

Select **1** in main menu to create a type. Then, give an input in the following format:

<type_name> <N=field_count> <field1_name> ... <fieldN_name>

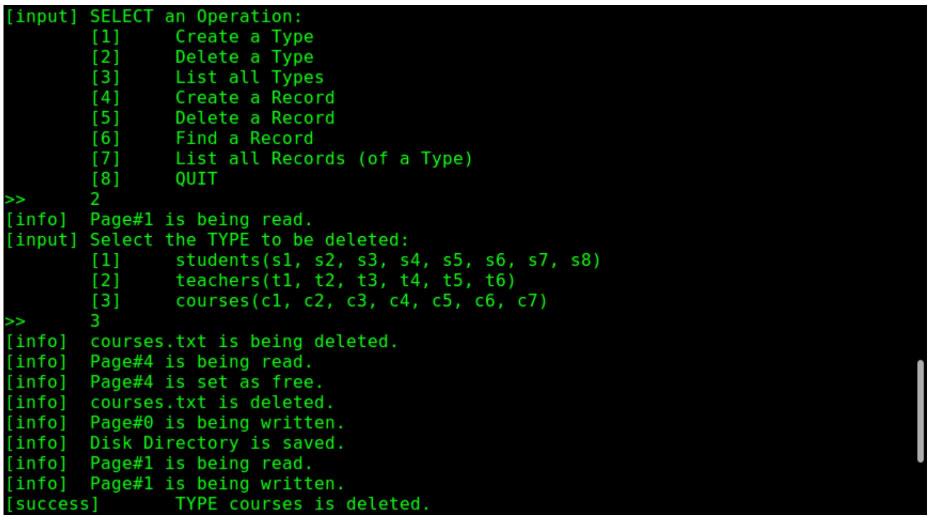


```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      1
[input] <type_name> <N=field_count> <filed1_name> <field2_name> ... <filedN_name>:
>>      students 8 s1 s2 s3 s4 s5 s6 s7 s8
[info]  Page#2 is being read.
[info]  Page#2 is being written.
[info]  New page (Page#2) is allocated.
[info]  students.txt is created.
[info]  Page#0 is being written.
[info]  Disk Directory is saved.
[info]  Page#1 is being read.
[info]  Page#1 is being written.
[success]       New TYPE (students) is created.
```

A new file for the type is created and the first free page (in this case, Page#2) on disk is allocated to it. Then, **File Manager** updates Disk Directory (which resides on Page#0). Then, System Catalog is updated (which resides only on Page#1 in this example).

3.2 Deleting a Type

Select **2** in main menu to delete a type. Then, select the TYPE to be deleted.



```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      2
[info]  Page#1 is being read.
[input] Select the TYPE to be deleted:
        [1]     students(s1, s2, s3, s4, s5, s6, s7, s8)
        [2]     teachers(t1, t2, t3, t4, t5, t6)
        [3]     courses(c1, c2, c3, c4, c5, c6, c7)
>>      3
[info]  courses.txt is being deleted.
[info]  Page#4 is being read.
[info]  Page#4 is set as free.
[info]  courses.txt is deleted.
[info]  Page#0 is being written.
[info]  Disk Directory is saved.
[info]  Page#1 is being read.
[info]  Page#1 is being written.
[success]       TYPE courses is deleted.
```

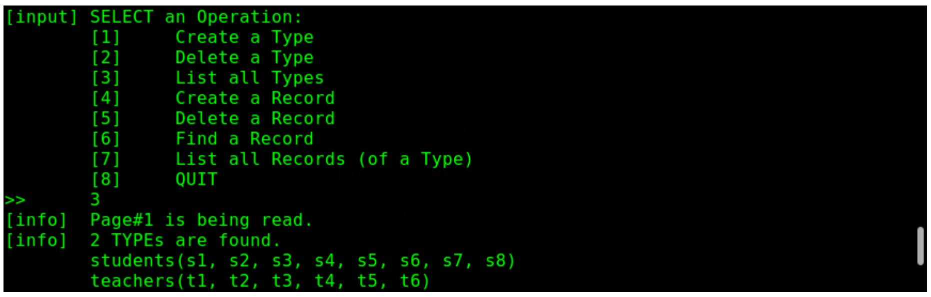
Suppose that 2 more types have been created after **students** in section **3.1** as follows:

- **teachers**: Page#3 is allocated for teachers.txt
- **courses**: Page#4 is allocated for courses.txt

For TYPE listing, System Catalog (which resides only on Page#1 in this example) is read. When a type is deleted, first its file is deleted from the disk and all the pages of that file are set as free. In this example, **courses.txt** consists of only Page#4. Then, **File Manager** updates Disk Directory (which resides on Page#0). Then, the type is deleted from System Catalog (which resides only on Page#1 in this example).

3.3 Listing All Types

Select **3** in main menu to list all the types.



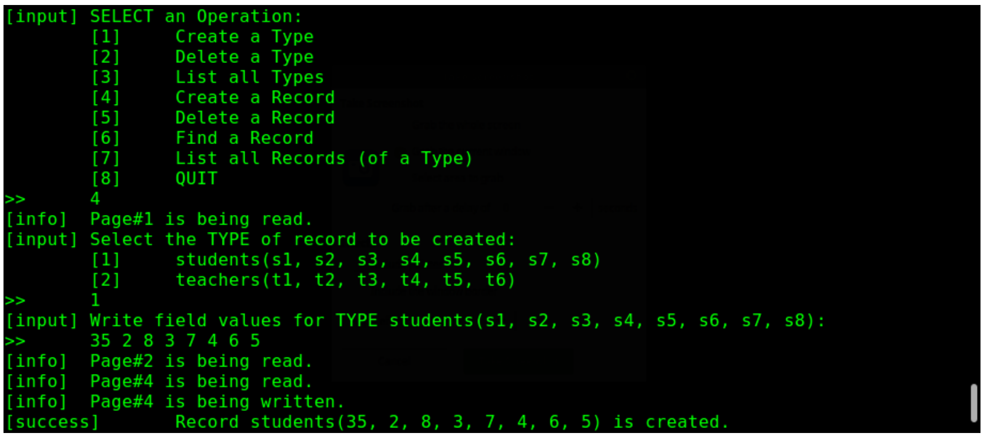
```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      3
[info]  Page#1 is being read.
[info]  2 TYPEs are found.
        students(s1, s2, s3, s4, s5, s6, s7, s8)
        teachers(t1, t2, t3, t4, t5, t6)
```

All the pages of System Catalog (which resides only on Page#1 in this example) file are read one by one and data types are collected. Then, the result is printed.

3.4 Creating a Record

Select **4** in main menu to create a record. Then, select the TYPE of record to be created. Then, enter the values of fields in the following format:

<field1_value> <field2_value> ... <fieldN_value>



```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      4
[info]  Page#1 is being read.
[input] Select the TYPE of record to be created:
        [1]     students(s1, s2, s3, s4, s5, s6, s7, s8)
        [2]     teachers(t1, t2, t3, t4, t5, t6)
>>      1
[input] Write field values for TYPE students(s1, s2, s3, s4, s5, s6, s7, s8):
>>      35 2 8 3 7 4 6 5
[info]  Page#2 is being read.
[info]  Page#4 is being read.
[info]  Page#4 is being written.
[success]       Record students(35, 2, 8, 3, 7, 4, 6, 5) is created.
```

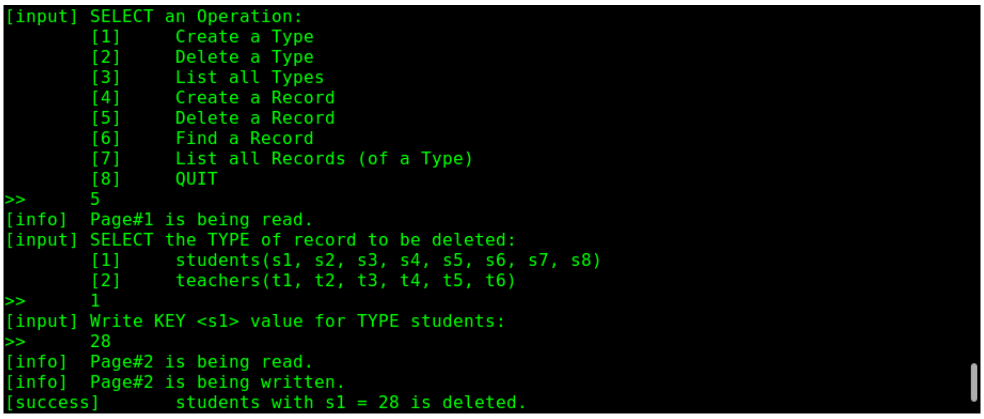
Suppose, 34 records of TYPE(students) have been created with values of the first fields *1, 2, ..., 34*. Since TYPE(students) has 8 fields, a page can contain up to **31** records of this type. The first 31 records reside on Page#2 and the last 3 records reside on Page#4. (Page#3 is allocated for TYPE(teachers))

For TYPE listing, System Catalog (which resides only on Page#1 in this example) is read. Then, pages of **student.txt** (Page#2 and Page#4) are read by one by until an empty space for the record is found. In this example, the second page of **students.txt** (page#4) is empty.

Note that the first field of a record is considered as the (primary) key.

3.5 Deleting a Record

Select **5** in main menu to delete a record. Then, select the TYPE of record to be deleted. Then, enter the (primary) key of the record.



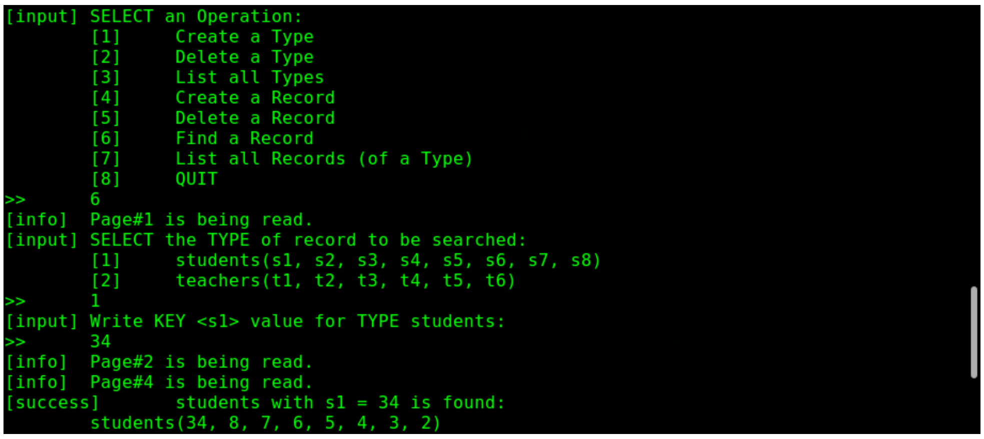
```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      5
[info]  Page#1 is being read.
[input] SELECT the TYPE of record to be deleted:
        [1]     students(s1, s2, s3, s4, s5, s6, s7, s8)
        [2]     teachers(t1, t2, t3, t4, t5, t6)
>>      1
[input] Write KEY <s1> value for TYPE students:
>>      28
[info]  Page#2 is being read.
[info]  Page#2 is being written.
[success]       students with s1 = 28 is deleted.
```

After section **3.4**, there are 35 records of TYPE(students). First 31 records resides on Page#2 and last 4 records are reside on Page#4. (Page#3 is allocated for TYPE(teachers))

For TYPE listing, System Catalog (which resides only on Page#1 in this example) is read. Then, record to be deleted is searched in pages of **students.txt** page by page. In this example, students(s1 = 28) resides on the first page (Page#2) of **students.txt**.

3.6 Searching for a Record

Select **6** in main menu to search for a record. Then, select the TYPE of record to be found. Then, enter the (primary) key of the record.

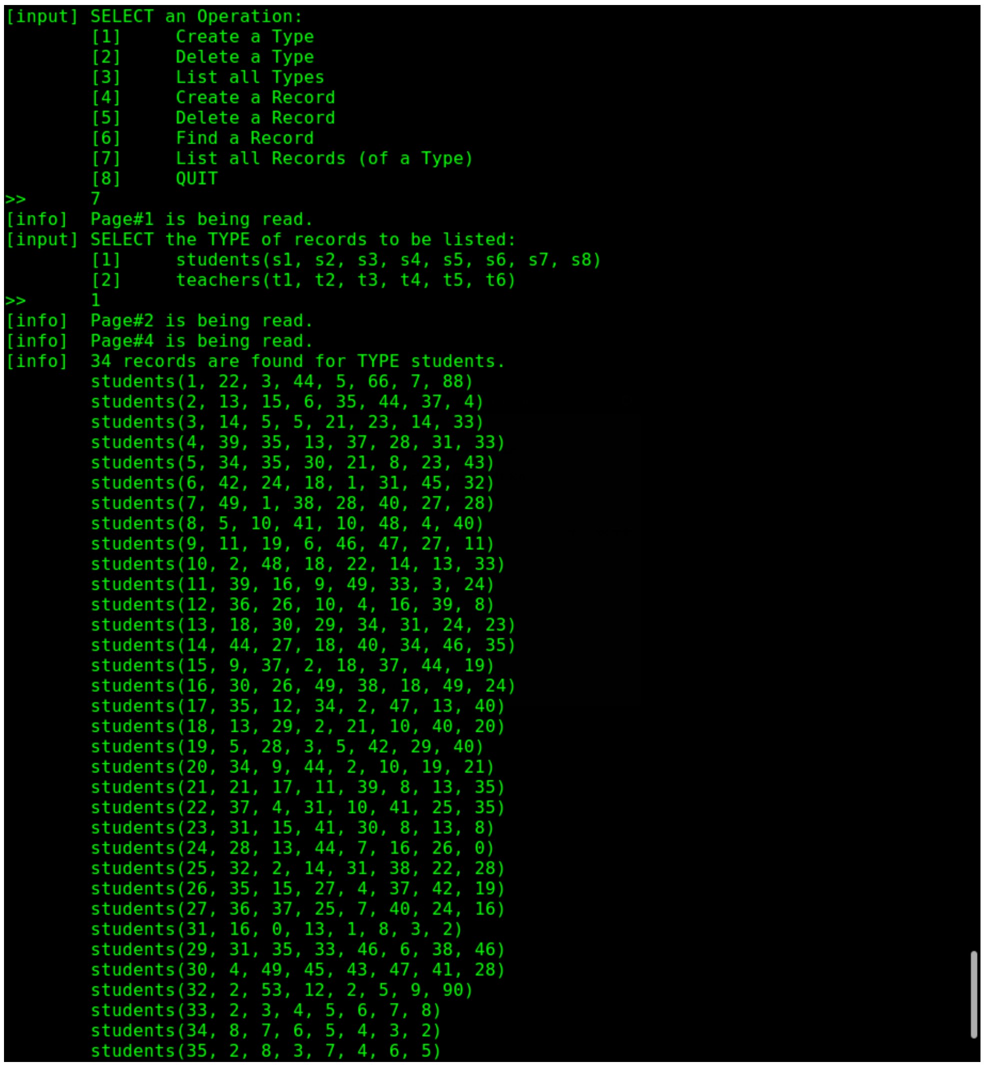


```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      6
[info]  Page#1 is being read.
[input] SELECT the TYPE of record to be searched:
        [1]     students(s1, s2, s3, s4, s5, s6, s7, s8)
        [2]     teachers(t1, t2, t3, t4, t5, t6)
>>      1
[input] Write KEY <s1> value for TYPE students:
>>      34
[info]  Page#2 is being read.
[info]  Page#4 is being read.
[success]       students with s1 = 34 is found:
        students(34, 8, 7, 6, 5, 4, 3, 2)
```

For TYPE listing, System Catalog (which resides only on Page#1 in this example) is read. Then record is searched in pages of **students.txt** page by page. In this example students(s1 = 34) resides on the second page (Page#4) of **students.txt**.

3.7 Listing All Records of a Type

Select **7** in main menu to list all records of a type. Then, select the TYPE of records to be listed.



```
[input] SELECT an Operation:
        [1]     Create a Type
        [2]     Delete a Type
        [3]     List all Types
        [4]     Create a Record
        [5]     Delete a Record
        [6]     Find a Record
        [7]     List all Records (of a Type)
        [8]     QUIT
>>      7
[info]  Page#1 is being read.
[input] SELECT the TYPE of records to be listed:
        [1]     students(s1, s2, s3, s4, s5, s6, s7, s8)
        [2]     teachers(t1, t2, t3, t4, t5, t6)
>>      1
[info]  Page#2 is being read.
[info]  Page#4 is being read.
[info]  34 records are found for TYPE students.
        students(1, 22, 3, 44, 5, 66, 7, 88)
        students(2, 13, 15, 6, 35, 44, 37, 4)
        students(3, 14, 5, 5, 21, 23, 14, 33)
        students(4, 39, 35, 13, 37, 28, 31, 33)
        students(5, 34, 35, 30, 21, 8, 23, 43)
        students(6, 42, 24, 18, 1, 31, 45, 32)
        students(7, 49, 1, 38, 28, 40, 27, 28)
        students(8, 5, 10, 41, 10, 48, 4, 40)
        students(9, 11, 19, 6, 46, 47, 27, 11)
        students(10, 2, 48, 18, 22, 14, 13, 33)
        students(11, 39, 16, 9, 49, 33, 3, 24)
        students(12, 36, 26, 10, 4, 16, 39, 8)
        students(13, 18, 30, 29, 34, 31, 24, 23)
        students(14, 44, 27, 18, 40, 34, 46, 35)
        students(15, 9, 37, 2, 18, 37, 44, 19)
        students(16, 30, 26, 49, 38, 18, 49, 24)
        students(17, 35, 12, 34, 2, 47, 13, 40)
        students(18, 13, 29, 2, 21, 10, 40, 20)
        students(19, 5, 28, 3, 5, 42, 29, 40)
        students(20, 34, 9, 44, 2, 10, 19, 21)
        students(21, 21, 17, 11, 39, 8, 13, 35)
        students(22, 37, 4, 31, 10, 41, 25, 35)
        students(23, 31, 15, 41, 30, 8, 13, 8)
        students(24, 28, 13, 44, 7, 16, 26, 0)
        students(25, 32, 2, 14, 31, 38, 22, 28)
        students(26, 35, 15, 27, 4, 37, 42, 19)
        students(27, 36, 37, 25, 7, 40, 24, 16)
        students(31, 16, 0, 13, 1, 8, 3, 2)
        students(29, 31, 35, 33, 46, 6, 38, 46)
        students(30, 4, 49, 45, 43, 47, 41, 28)
        students(32, 2, 53, 12, 2, 5, 9, 90)
        students(33, 2, 3, 4, 5, 6, 7, 8)
        students(34, 8, 7, 6, 5, 4, 3, 2)
        students(35, 2, 8, 3, 7, 4, 6, 5)
```

For TYPE listing, System Catalog (which resides only on Page#1 in this example) is read. Then, all the pages of **students.txt** (Page#2 and Page#4) are read and records in them are collected. Then the result is printed.

Note that when the record students(s1 = 28) is deleted in section **3.5**, the last record (students(s1 = 31)) of the related page (Page#2) is moved into the position of students(s1 = 28).

4 CONCLUSIONS AND ASSESSMENT

In this assignment, I implemented a simple Storage Manager assuming the user always provides valid inputs and hence not doing any error checking. Since I didn't change my initial design except for System Catalog file structure, ups and downs of my design are the same as in the previous assignment. To recap some important issues:

I kept structures fixed-size which makes them easy to maintain and operate on. This has a negative impact on disk usage. This impact is most evident in System Catalog file in which every record is forced to take 146 bytes even if it can need much less space. *(See documentation of Assignment 1)*

In addition, I implemented my design so that once a page is allocated to a file, it remains allocated to it even if at some point all the records in it are deleted. Pages are de-allocated only when the user deletes the related file. This situation can be severe in terms of disk usage inefficiency.

Lack of structures such as indexes in my design makes searching for a record a slow operation. However, since records are inserted linearly to pages, creating and listing all records are fast. Deletion is also fast once the record to be deleted is found.