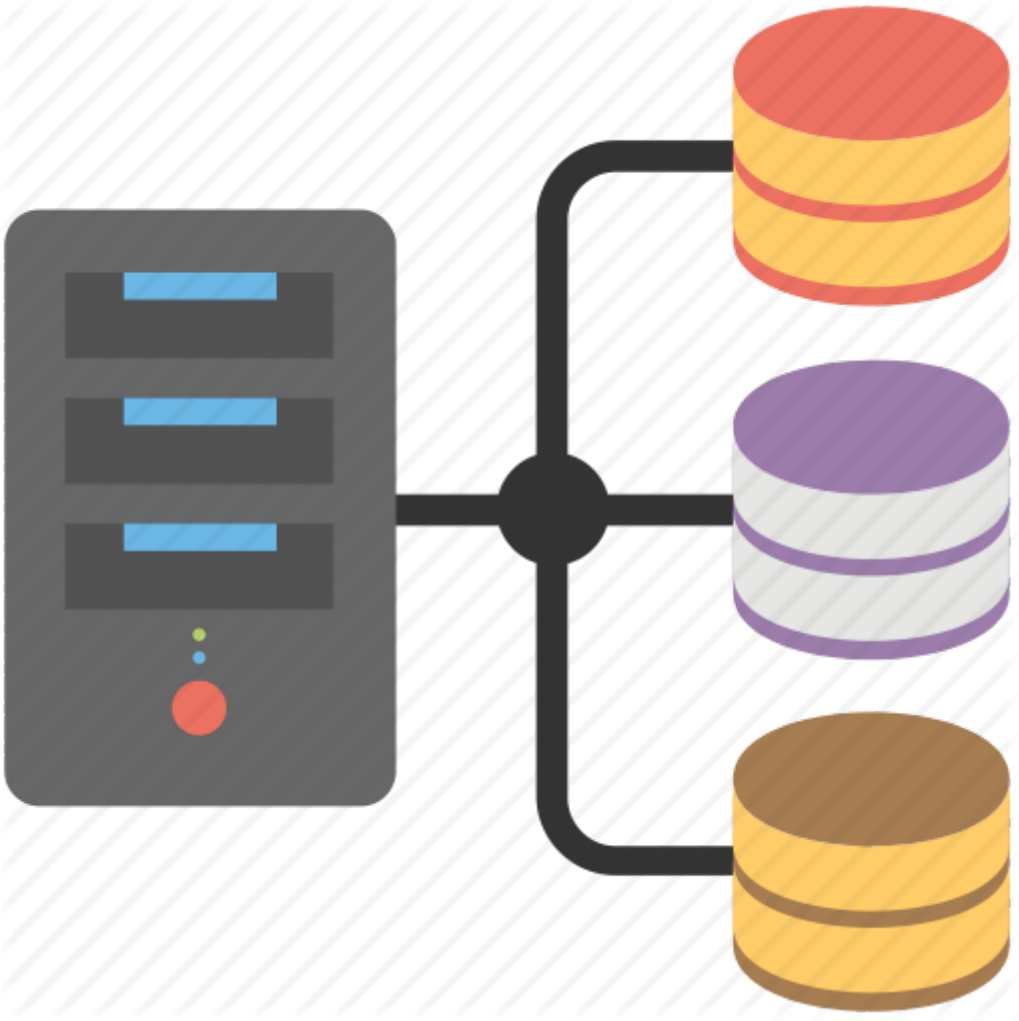
CmpE 321 - 2020/Summer

Project - 1:

Simple Storage Management System



Prepared by:

Hazar ÇAKIR - 2017400093

I) Introduction:

This is a design project that is actually the first part of the greater project. We are expected to design a storage management system basically. System should fulfill some basic type operations such as creating a type, deleting a type and list all types and some record operations such as creating a record, deleting a record, searching a record and listing all records of a type. While designing such a system, there will be some assumptions and constraints which will be described in detail in the next chapter. Operations will be explained as pseudocode with comments in english.

In the operation section, firstly I describe all processes in English, then I realized that this is not the preferred way, I changed my design to a more code-like way. However, I decided to keep English sentences too. So there is both English explanation and pseudocode thereafter.

II) Assumptions & Constraints:

a) Numeral constraints:

- Page Size: 2 KBs
- File Size: Every file can have unlimited pages.
- Max number of fields that a type can have: 32
- Min number of fields that a type can have: 3
- Max length of a type name: 20 characters
- Max length of a field name: 20 characters
- Min length of a type name: 8 characters
- Min length of a field name: 8 characters
- Size of a field: 4 bytes
- Value range of a field: -2^31 --- 2^31 -1.

b) Assumptions:

- User always enters valid input.
- It is assumed that valid input can violate boundary condition and should be handled in the program.
- All fields are integers. However, type and field names are alphanumeric.
- A disk manager already exists that is able to retrieve a page when its address is given.
- A character’s size is 1 byte.

- First field of a record is accepted as the primary key.
- Every type name should be unique and DDL operations will be done with type names.
- Every primary key is unique.
- Every field of a record has the same size disregarding the data inside which is signed integer (4 bytes) that allows store data from -2^31 to 2^31 -1.
- Every record of a type has the same size calculated from the number of fields that that record has. This is because when a record has less fields, a page can store more of that type records and that would decrease page accesses. Calculation done as field size times number of fields plus record header's size and this data stored in the system catalogue. Explained in detail in the next chapter.
- Page header consist of these fields:
    - Full flag (1 or 0)
    - Pointer_to_the_Next_Page
    - Number_of_Records
    - Max_Number_of_Records
- Record header consist of these fields:
    - Deletion_bit (1 or 0)
- It is assumed that the File Manager can request System Catalogue by Disk Manager directly and Disk Manager returns the System Catalogue.
- It is assumed that a garbage collector cleans the file which setted **deleted** from the user in the System Catalogue.

III) Storage Structures:

a) System Catalogue:

<table>
<tr><th colspan="11">System Catalogue</th></tr>
<tr><th>1 byte</th><th>20 bytes</th><th>...</th><th>1 byte</th><th colspan="6">20 bytes each field name (32 * 20 = 640 bytes)</th><th>1 byte</th></tr>
<tr><th>Deleted</th><th>Type Name</th><th>Pointer to_File's First Page</th><th>Number of Fields</th><th colspan="6">Name_of_Fields</th><th>Size_of a_Record</th></tr>
<tr><td>0-1</td><td>Type name 1</td><td>*</td><td>n</td><td>Field 1</td><td>Field 2</td><td>Field 3</td><td>...</td><td>Field n-1</td><td>Field n</td><td>4*n + 1 bytes</td></tr>
</table>

0-1	Type name 2	*	15	Field' 1	Field' 2	Field' 3	...	Field' 15	Empty...	61 bytes
0-1	Type name 3	*	2	Field'' 1	Field'' 2	Empty	...	Empty	Empty	9 bytes
0-1	Type name 4	*	6	Field''' 1	Field''' 2	Field''' 3	...	Empty	Empty	25 bytes
...	...	...	...	...	...	...	...	...	...	..

System catalogue has no header and the structure of the metadata in it is described in the chart. There is:

- **Deleted** bit: 1 when a type is deleted, 0 otherwise and normally.
- **Type_name**: Max 20 character long type name
- **Pointer_to_File's_First_Page**: Stores the address of the beginning of the file.
- **Number_of_Fields**: Determined by the user when creating a type.
- **Name_of_Fields**: There is a place for 32 fields which is max number of fields. First **Number_of_Fields** place is filled with field names determined by the user.
- **Size_of_a_Record:** Calculated by **Number_of_Fields * Size_of_a_Field + Size_of_Record_Header** which is **Number_of_Fields * 4 + 1** bytes.

b) Page Header and Page Structure:

Page Header				Rest of the Page		
1 byte	**...**	**2 byte**	**2 byte**			
Full	**Pointer_to the_Next Page**	**Number_of Records**	**Max_Number_of Records**			
1-0	*	15	200	Record 1	Record 2	...

Page header has 4 fields to store some specific data which are:

- **Full** bit: This is 1 when the page is full. Determined by the **Number_of_Records** and **Max_Number_of_Records** fields. When these 2 are equal, Full bit set to 1, otherwise 0.
- **Pointer_to_the_Next_Page:** Stores the address of the next page.
- **Number_of_Records:** Number of the records in the page. Modified by the user when creating or deleting a record.
- **Max_Number_of_Records:** Calculated via **Size_of_a_Record** in the System Catalogue as **rounding down 2048 - Page_Header_Size** / **Size_of_a_Record** which is **rounding down 2043 / Size_of_a_ Record.** (The size of the **Pointer_to_the_Next_Page** is omitted because it depends on the systems addressing system and should be evaluated also in the real system)

c) Record Header and Record Structure:

Record Header	Rest of the Record			
1 byte	**4 bytes**	**4 bytes**		**4 bytes**
Deleted	**Field_1**	**Field_2**		**Field_n**
1-0	Data 1	Data 2	**...**	Data n

Record header has only one field which is Deletion_bit:

- **Deleted:** 1 when the record is deleted, 0 otherwise.

IV) Operations:

a) Create a type:

- **Inputs taken from user:**
  - **Type Name:** 8 to 20 character unique name
  - **Number of fields:** How many fields that that type has
  - **Names of fields:** Names of the files one by one

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager. File

Manager traverses the System Catalogue till find an empty place to store type and its fields name.

When finding a free place, set **Deleted** to **0,** and set **Type_Name, Number_of_Fields, Name_of_Fields** fields from input data. Calculate size of a record of that type and set **Size_of_a_Record** field to that value.

Then a new File and a new Page which is the first page of that file should be created. File Manager requests Disk Manager to retrieve the first non-allocated place to create a new file and a page. The location of the first page of the file stored in the **Pointer_to_File's_First_Page** field.

This new page's header is filled at the creation of the page. **Full** fields sets **0, Pointer_to_the_Next_Page** is null for that time, **Number_of_Records** sets **0** and **Max_Number_of_Records** is calculated as described before from the data in System Catalogue and assigned.

```
function createType(){
    open(System_Catalogue)
    // File Manager requests Disk Manager to retrieve System Catalogue. Disk
    // Manager retrieves System Catalogue and passes it to the File Manager.

    takeInput(Type_name)
    if(length(Type_name) <8 or length(Type_name)>20 or Type_name is in
                                                    System_Catalogue){
        return invalid_Type_name_error
    }endif

    takeInput(Number_of_fields)
    if ( Number_of_fields < 3 or Number_of_fields > 32){
        return invalid_Number_of_fields_error
    }endif
    for( count = 0 → Number_of_fields){
        takeInput(Name_of_field)
        if(length(Name_of_field) <8 or length(Name_of_field)>20 ){
            return invalid_field_name_error
        }endif
    }endif
    }endfor

    for( line in System_Catalogue )
        if( line is empty ){
            break
        } endif
    }endfor
```

// File Manager traverses the System Catalogue till find an empty place to
// store type and its fields name.
line_add(deleted_value) // which is 0
line_add(Type_Name **)**

createFile(File_of_type_Name **)**
line_add(pointer_to_file_of_type_Name**)**
// A new File and a new Page which is the first page of that file should be
// created. File Manager requests Disk Manager to retrieve the first
// non-allocated place to create a new file and a page. The location of the first
// page of the file stored in the **Pointer_to_File's_First_Page** field.

line_add(Number_of_fields **)**
line_add(Names_of_field **)**
calculate(Size_of_a_Record **) //** which is **Number_of_Fields * 4 + 1**
line_add(Size_of_a_Record **)**

open(File_of_type_Name **)**
header_add(full_value **) //** which is 0
header_add(Pointer_to_the_Next_Page **) //** which is null
header_add(Number_of_Records **) //** which is 0
calculate(Max_Number_of_Records **) //** which is **round down 2043 / // Size_of_a_ Record.**
header_add(Max_Number_of_Records **)**

// This new page's header is filled at the creation of the page. **Full** fields sets
// **0, Pointer_to_the_Next_Page** is null for that time, **Number_of_Records**
// sets **0** and **Max_Number_of_Records** is calculated as described before
// from the data in System Catalogue and assigned.

}endfunction

b) Delete a type:

- **Inputs taken from user:**
  - **Type Name:** Type name that will be deleted

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue till find the same Type name field value as Type Name that is taken from the user to delete. When finding the corresponding type, its **Deleted** field setted to 1. If there is no type as the same as input, return an error to the user.

```
function deleteType(){
    open(System_Catalogue)
    // File Manager requests Disk Manager to retrieve System Catalogue. Disk
    // Manager retrieves System Catalogue and passes it to the File Manager.

    takeInput(Type_name)
    if(Type_name is not in System_Catalogue){
        return invalid_Type_name_error
    }endif


    for( line in System_Catalogue )
        if( line[Type_Name] == input_Type_name  ){
            break
        } endif
    }endfor
    // File Manager traverses the System Catalogue till find the same Type name
    // field value as Type Name that is taken from the user to delete.
    line[deleted_value] = 1

}endfunction
```

c) List all types:

- **Inputs taken from user:**
  - **No information needed**

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue and accumulates type names in a list. When reaching the non-allocated place, it returns the list of the types.

```
function listAllTypes(){
    open(System_Catalogue)
```

```
        // File Manager requests Disk Manager to retrieve System Catalogue. Disk
        // Manager retrieves System Catalogue and passes it to the File Manager.

        list_of_types = [ ]

        for( line in System_Catalogue )
                list_of_types.add( line[Type_name] )
        }endfor
        // File Manager traverses the System Catalogue and accumulates type names
        // in a list. When reaching the non-allocated place, it stops
        return list_of_types
}endfunction
```

d) Create a record:

- **Inputs taken from user:**
  - **Type Name:** Type name of the record.
  - **Field Values:** Regarding the number of fields, user should give that number of integer values.

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue till find the Type Name that is taken from the user to create a record. When finding the corresponding type, we retrieve **Pointer_to_File's_First_Page** field.

The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_File's_First_Page** field. The disk manager retrieves the page and passes it to the file manager.

We look at the Page Header's **Full** field. If it is **0,** that means the page has a place to store at least one more record. File manager traverses the page record by record and Storage Manager checks every record's header's **Deleted** field if it is **1.** If there is a record whose header's **Deleted** field's value is 1, new information is written over that record. If there is no record in that type and File Manager reaches free space, a new Record is created and stored in that place. New Records header's **Deleted** field set to **0** and field values stored after header.

After inserting a new record, we retrieve Page Header's **Number_of_Records** field and increment its value. After incrementation, we control whether **Number_of_Records** fields value is equal to **Max_Number_of_Records** fields value. If there is no equality, creation of a record is done. If they are equal, Page Header's **Full** field is set to 1 and File Manager requests Disk Manager to

retrieve the first non-allocated place to create a new page. The location of the page stored in the Page Header's **Pointer_to_the_Next_Page** field.

New Page's Header fields initialized in the creation same as the creation of the first page of a type.

In the other case,if the Page Header's **Full** field is **1,** we retrieve the page header's **Pointer_to_the_Next_Page** field. The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_the_Next_Page** field. The disk manager retrieves the page and passes it to the file manager. And same controls are done till find a page that has **Full** field as **0.** In that case, previous operations are done.

```
function createRecord(){
    open(System_Catalogue)
    // File Manager requests Disk Manager to retrieve System Catalogue. Disk
    // Manager retrieves System Catalogue and passes it to the File Manager.

    takeInput(Type_name)
    if(Type_name is not in System_Catalogue){
        return invalid_Type_name_error
    }endif
    for( line in System_Catalogue )
        if( line[Type_Name] == input_Type_name  ){
            break
        } endif
    }endfor
    // File Manager traverses the System Catalogue till find the Type Name that is
    // taken from the user to create a record.

    open_file_from_pointer( line[ pointer_to_file_of_type_Name ] )
    // The file manager requests the disk manager to retrieve the page with the
    // address obtained from Pointer_to_File's_First_Page field. The disk
    // manager retrieves the page and passes it to the file manager.

    while( header[Full] == 1 ){
        close( previous_file )
        open_page_from_pointer( header[ Pointer_to_the_Next_Page ] )
    }endWhile
    // while the Page Header's Full field is 1, we retrieve the page header's
    // Pointer_to_the_Next_Page field. The file manager requests the disk
    // manager to retrieve the page with the address obtained from
    // Pointer_to_the_Next_Page field. The disk manager retrieves the page and
```

// passes it to the file manager. And same controls are done till find a page
// that has **Full** field as **0.**

tempRecord
for(record **in** page)
    if(record[deleted] == 1){
        tempRecord = record
        break
    } endif
}endfor
if(tempRecord == null **){**
    page_add(newRecord **)**
newRecord[deleted] = 0
for(count = 0 → number_of_fields **)**
    take_input(field_value **)**
    if(field_value > 2^31 -1 **or** field_value < -2^31){
        return field_size_error
    }endif
    newRecord_add(field_value)
}endfor
// File manager traverses the page record by record and Storage Manager
// checks every record's header's **Deleted** field if it is **1.** If there is a record
// whose header's **Deleted** field's value is 1, new information is written over
// that record. If there is no record in that type and File Manager reaches free
// space, a new Record is created and stored in that place. New Records
// header's **Deleted** field set to **0** and field values stored after header.

header**[** Number_of_Records **] =** header**[** Number_of_Records **] + 1**
if(header[Number_of_Records **] ==** header[Max_Number_of_Records]){
    createPage(Next_page **)**
    header[Pointer_to_the_Next_Page **] =** pointer_of_next_page
    open(Next_page **)**
    header_add(full_value **) //** which is 0
    header_add(Pointer_to_the_Next_Page **) //** which is null
    header_add(Number_of_Records **) //** which is 0
    calculate(Max_Number_of_Records **) //** which is **round down 2043 /**
    // **Size_of_a_ Record.**
    header_add(Max_Number_of_Records **)**
}endif
// After inserting a new record, we retrieve Page Header's
//**Number_of_Records** field and increment its value. After incrementation, we
//control whether **Number_of_Records** fields value is equal to

//**Max_Number_of_Records** fields value. If there is no equality, creation of a //record is done. If they are equal, Page Header's **Full** field is set to 1 and File //Manager requests Disk Manager to retrieve the first non-allocated place to //create a new page. The location of the page stored in the Page Header's //**Pointer_to_the_Next_Page** field.New Page's Header fields initialized in the //creation same as the creation of the first page of a type.

}endfunction

e) Delete a record:

- **Inputs taken from user:**
  - **Type Name:** Type name pf the report that will be deleted
  - **Key Field Value:** Value of the first field of the record that will be deleted which is primary key of record

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue till find the Type Name that is taken from the user to delete a record. When finding the corresponding type, we retrieve **Pointer_to_File's_First_Page** field.

The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_File's_First_Page** field. The disk manager retrieves the page and passes it to the file manager.

File manager traverses the page record by record and Storage Manager checks every record's first field which is the primary key, if it is the same as with the input value**.** When we found the target record, just set its header's **Deleted** field to **1.** After deletion, we retrieve the page header's **Number_of_Records** field and decrement it.

If there is no record which has the same key value as input, we take a look at the page header's **Full** field if it is **1.**

In that case, we retrieve the page header's **Pointer_to_the_Next_Page** field. The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_the_Next_Page** field. The disk manager retrieves the page and passes it to the file manager. And same controls are done till find corresponding record. Again after deletion, we retrieve the page header's **Number_of_Records** field and decrement it.

If the page header's **Full** field is **0,** that means there is no record to delete. We return an error to the user.

function deleteRecord(){

open(System_Catalogue**)**

// File Manager requests Disk Manager to retrieve System Catalogue. Disk
// Manager retrieves System Catalogue and passes it to the File Manager.

takeInput(Type_name**)**
takeInput(records_first_field **)**
if(Type_name **is not in** System_Catalogue**){**
    return invalid_Type_name_error
}endif

for(line **in** System_Catalogue)
    if(line[Type_Name] == input_Type_name){
        break
    } endif
}endfor
// File Manager traverses the System Catalogue till find the Type Name that is
// taken from the user to create a record.

open_file_from_pointer(line[pointer_to_file_of_type_Name] **)**
// The file manager requests the disk manager to retrieve the page with the
// address obtained from **Pointer_to_File's_First_Page** field. The disk
// manager retrieves the page and passes it to the file manager.

while(true){
    for(record **in** activePage)
        if(record[field1] == input_record_first_field){
            record[deleted] = 1
            header**[** Number_of_Records **] =** header**[**Number_of_Records **] - 1**
            header[Full] = 0
            return true
        } endif
        //File manager traverses the page record by record and Storage
        //Manager checks every record's first field which is the primary
        //key, if it is same as with the input value**.** When we found the
        //target record, just set its header's **Deleted** field to **1.** After
        //deletion, we retrieve the page header's **Number_of_Records**
        //field and decrement it.
    }endfor
    if (header[Full **] == 0) {**
        return error
    // If there is no record which has the same key value as input, we take
    a look at the page header's **Full** field if it is **1.** If the page header's **Full**

field is **0,** that means there is no record to delete. We return an error to the user.

} else {

close(previous_file **)**

open(header[Pointer_to_the_Next_Page **])**

// If the page header's **Full** field is **1,** we retrieve the page header's //**Pointer_to_the_Next_Page** field. The file manager requests the disk //manager to retrieve the page with the address obtained from //**Pointer_to_the_Next_Page** field. The disk manager retrieves the //page and passes it to the file manager.

}endif

}endWhile

}endfunction

f) Search for a record:

- **Inputs taken from user:**
  - **Type Name:** Type name pf the report that is searched
  - **Key Field Value:** Value of the first field of the record that is searched which is primary key of record

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue till find the Type Name that is taken from the user to delete a record. When finding the corresponding type, we retrieve **Pointer_to_File's_First_Page** field.

The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_File's_First_Page** field. The disk manager retrieves the page and passes it to the file manager.

File manager traverses the page record by record and Storage Manager checks every record's first field which is the primary key, if it is same as with the input value**.** When we found the target record, return this record to the user**.** If there is no record which has the same key value as input, we take a look at the page header's **Full** field if it is **1.**

In that case, we retrieve the page header's **Pointer_to_the_Next_Page** field. The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_the_Next_Page** field. The disk manager retrieves the page and passes it to the file manager. And same controls are done till find corresponding record. If the page header's **Full** field is **0,** that means there is no record which fits input value. We return an error to the user.

```
function searchRecord(){
    open(System_Catalogue)
    // File Manager requests Disk Manager to retrieve System Catalogue. Disk
    // Manager retrieves System Catalogue and passes it to the File Manager.

    takeInput(Type_name)
    takeInput( records_first_field )
    if(Type_name is not in System_Catalogue){
        return invalid_Type_name_error
    }endif

    for( line in System_Catalogue )
        if( line[Type_Name] == input_Type_name ){
            break
        } endif
    }endfor
    // File Manager traverses the System Catalogue till find the Type Name that is
    // taken from the user to create a record.

    open_file_from_pointer( line[ pointer_to_file_of_type_Name ] )
    // The file manager requests the disk manager to retrieve the page with the
    // address obtained from Pointer_to_File's_First_Page field. The disk
    // manager retrieves the page and passes it to the file manager.

    while( true ){
        for( record in page)
            if( record[field1] == input_record_first_field ){
                return record
            } endif
        }endfor
        //File manager traverses the page record by record and Storage
        //Manager checks every record's first field which is the primary key, if it
        //is the same as with the input value. When we find the target record,
        //return this record to the user.

        if ( header[ Full ] == 0 ) {
            return not_found
        //If there is no record which has the same key value as input, we take a
        //look at the page header's Full field if it is 1. If the page header's Full
        //field is 0, that means there is no record to find. We return not_found to
        //the user.
        } else {
```

```
            close( previous_file )
            open( header[ Pointer_to_the_Next_Page ] )
        //If the page header's Full field is 1, we retrieve the page header's
        //Pointer_to_the_Next_Page field. The file manager requests the disk
        //manager to retrieve the page with the address obtained from
        //Pointer_to_the_Next_Page field. The disk manager retrieves the
        //page and passes it to the file manager. And same controls are done
        //till find corresponding record.

        }endif

    }endWhile
}endfunction
```

g) List all records of a type:

- **Inputs taken from user:**
  - **Type Name:** Type name pf the report that will be deleted

File Manager requests Disk Manager to retrieve System Catalogue. Disk Manager retrieves System Catalogue and passes it to the File Manager.

File Manager traverses the System Catalogue till find the Type Name that is taken from the user to delete a record. When finding the corresponding type, we retrieve **Pointer_to_File's_First_Page** field.

The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_File's_First_Page** field. The disk manager retrieves the page and passes it to the file manager.

File manager traverses the page record by record and accumulates every record. If the page header's **Full** field is **1,**we retrieve the page header's **Pointer_to_the_Next_Page** field after storing last in the page.

The file manager requests the disk manager to retrieve the page with the address obtained from **Pointer_to_the_Next_Page** field. The disk manager retrieves the page and passes it to the file manager. And same operations are done till find a page whose header's **Full** field is 0. After reaching the end of that file, it returns the list of all records.

```
function listAllRecords(){
    open(System_Catalogue)
    // File Manager requests Disk Manager to retrieve System Catalogue. Disk
    // Manager retrieves System Catalogue and passes it to the File Manager.
```

```
takeInput(Type_name)
list = [ ]
if(Type_name is not in System_Catalogue){
        return invalid_Type_name_error
}endif

for( line in System_Catalogue )
        if( line[Type_Name] == input_Type_name  ){
                break
        } endif
}endfor
// File Manager traverses the System Catalogue till find the Type Name that is
// taken from the user to create a record.

open_file_from_pointer( line[ pointer_to_file_of_type_Name ] )
// The file manager requests the disk manager to retrieve the page with the
// address obtained from Pointer_to_File's_First_Page field. The disk
// manager retrieves the page and passes it to the file manager.

while( true ){
        for( record in page)
                list.add( record )
        }endfor
        //File manager traverses the page record by record and accumulates
        //every record.
        if ( header[ Full  ] == 0 ) {
                return list
        // If the page header's Full field is 0, we return the list that we
        accumulate till now to the user.
        } else {
                close( previous_file )
                open( header[ Pointer_to_the_Next_Page  ] )
        //If the page header's Full field is 1,we retrieve the page header's
        //Pointer_to_the_Next_Page field after storing last in the page. The
        //file manager requests the disk manager to retrieve the page with the
        //address obtained from Pointer_to_the_Next_Page field. The disk
        //manager retrieves the page and passes it to the file manager.
        }endif

}endWhile
}endfunction
```

V) Conclusions & Assessment:

In the project, the relations between Store Manager, File Manager and Disk Manager described clearly. In all steps, the use of the storage structures is highly detailed. I designed the storage structures as efficiently as possible and I use all of the data fields in the operations. There is no redundant field. Some fields exist in order to increase the speed of the store manager.

I try to think as deep as possible and try to clarify all the important details. I hope the general structure is in prefered way

Hazar Çakır