CMPE 321 INTRODUCTION TO DATABASE SYSTEMS

SUMMER 2020

Project 1

Design of a Storage Management System

YAĞMUR SELEK
Id:2017400273

INTRODUCTION

We are asked to design a storage management system under some constraints and assumptions considering some details such as datas and users' input behaivours. Also we should make some assumptions regarding to sizes of files and pages or maximal length of some inputs and names can take.
Our design is expected to have to system catolog(data dictionary) that has a metadata and data storage units parts to store desired data.
Our system should be designed to satisfy following operations;
Data definition language operations such as create and delete a type and list all types.
Also, create and delete a record, search for a record and list all records of a type as data manipulation language operations.

ASSUMPTIONS

User always enters valid input. By that I assume that user not only enters the names, headers validly but also the operations that the user is going to perform will be as expected. Such as I assume user is not going to try deleting a non-existing type or record and not try to list the records for a non-existing type.
All fields are integers.
Type and field names are alphanumeric.
When an adress is given to retrieve a page a disk manager always exists.
Since only one type is recorded on a file I assume that a file has reasonable amount of pages.
I assume that user always enter minimum 3 and max 10 number of fields for a type.
I assume from the user that for both type names and field names one will enter from 8 and up to 24 alphanumeric characters.
All types has static number of fields and determined by user that how many field a type will have

CONSTRAINTS

Both type names and field names can have 8-24 alphanumeric characters.
A type can have a number of field from 3 up to 10.
Page size is 2KB
File size is 100MB at most

3.Storage Structures

3.1.SYSTEM CATALOGUE

System catalogue is the main part of my database management system since it stores metadata. Since the system catalog itself mainly represents as our System the system catalog is reachable anytime.
Basically the catalogue of the system stores:

- data types
- number of items that is being stored
- primary keys
- number of records
- number of pages

Arbitrarily my system catalog look like following diagram:

3.2 FILE

A file is a collection of records that consists in pages.
In my Database Management System every root file has a unique data type.
So that when the user enters the type the informations will be recorded to the page under that file. If the last page of that file is full than a new page is created under that file.
Files stored will be stored in doubly linked list in my system.

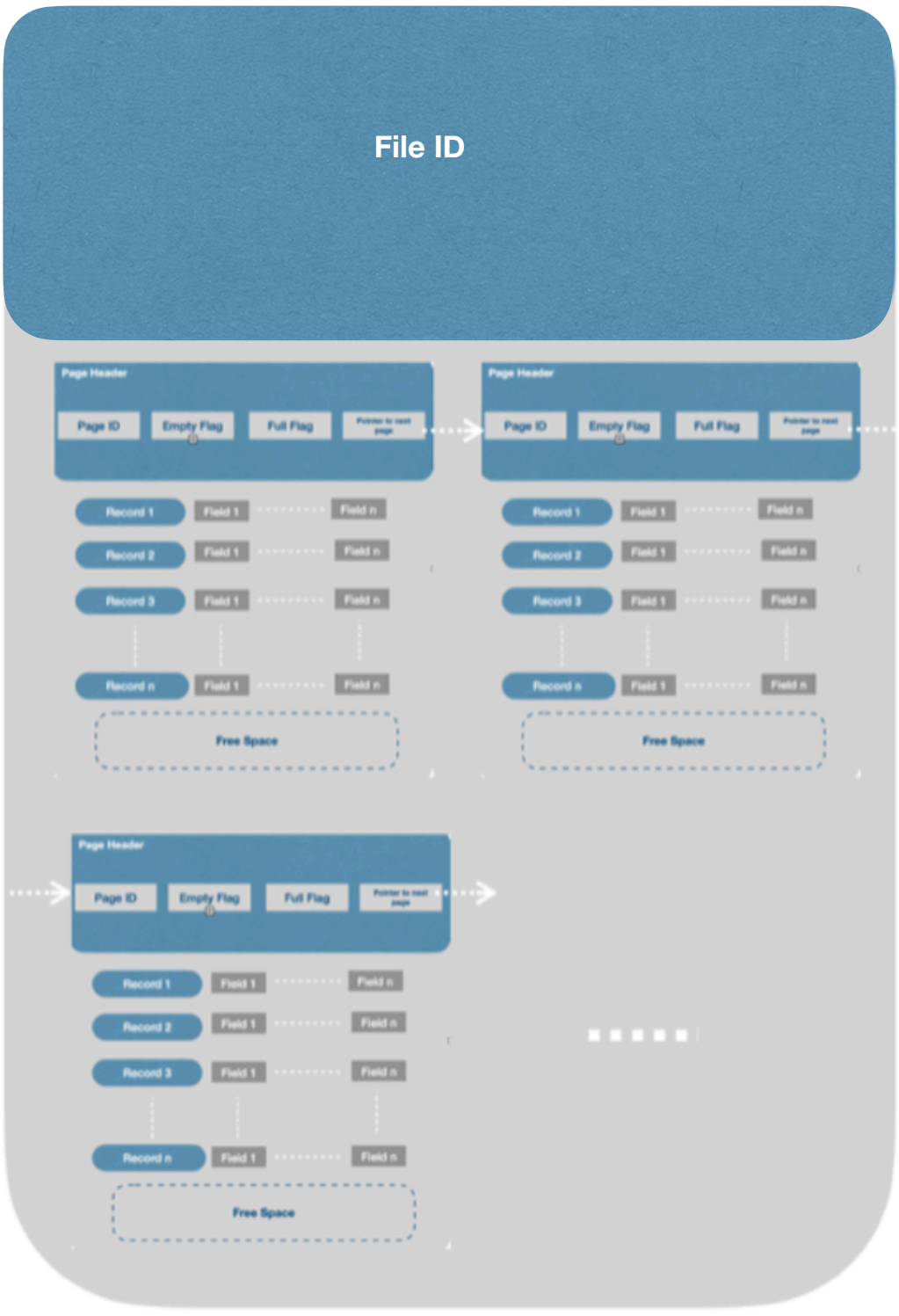

Diagram for a file

.Since I want a file has reasonable amount of pages(50000 at max). When file size reached its maximum storage another 2 files with same type will be created to store same type files has a tree like structure. Also we wont need to traverse all files when we find a root file for a type. Note that All root files still has a unique type.

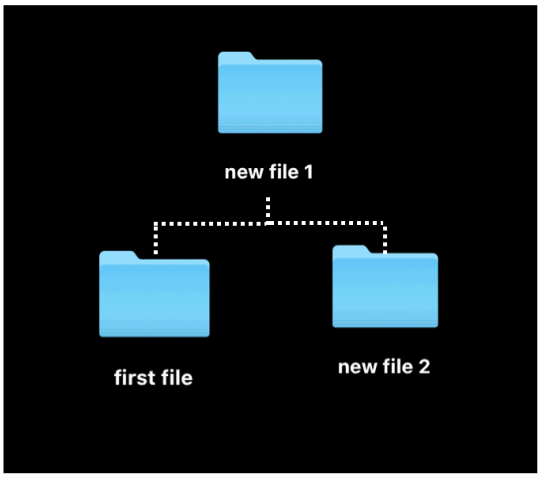

Diagram for same type of files created when needed

3.3 PAGE DESIGN

Page consists of page header and records and I wanted it to be as maximum size so that maximum number of records can be in one page. Thus according to constraints given my page is 2 KB it is reasonable as a page size and also maximum that it can be.

Page Header consists of followings:

- Page ID
- Empty flag (1 means empty page)
- Full flag (1 means full page)
- Pointer to next page. (Points null if it is the last page among the all pages in the file. If it is not full than **it points to next page in the file where the page itself exists**)
- Pointer to previous page.(Points null if it is the first page in that file)
- Type
- Number of records currently exists at that page

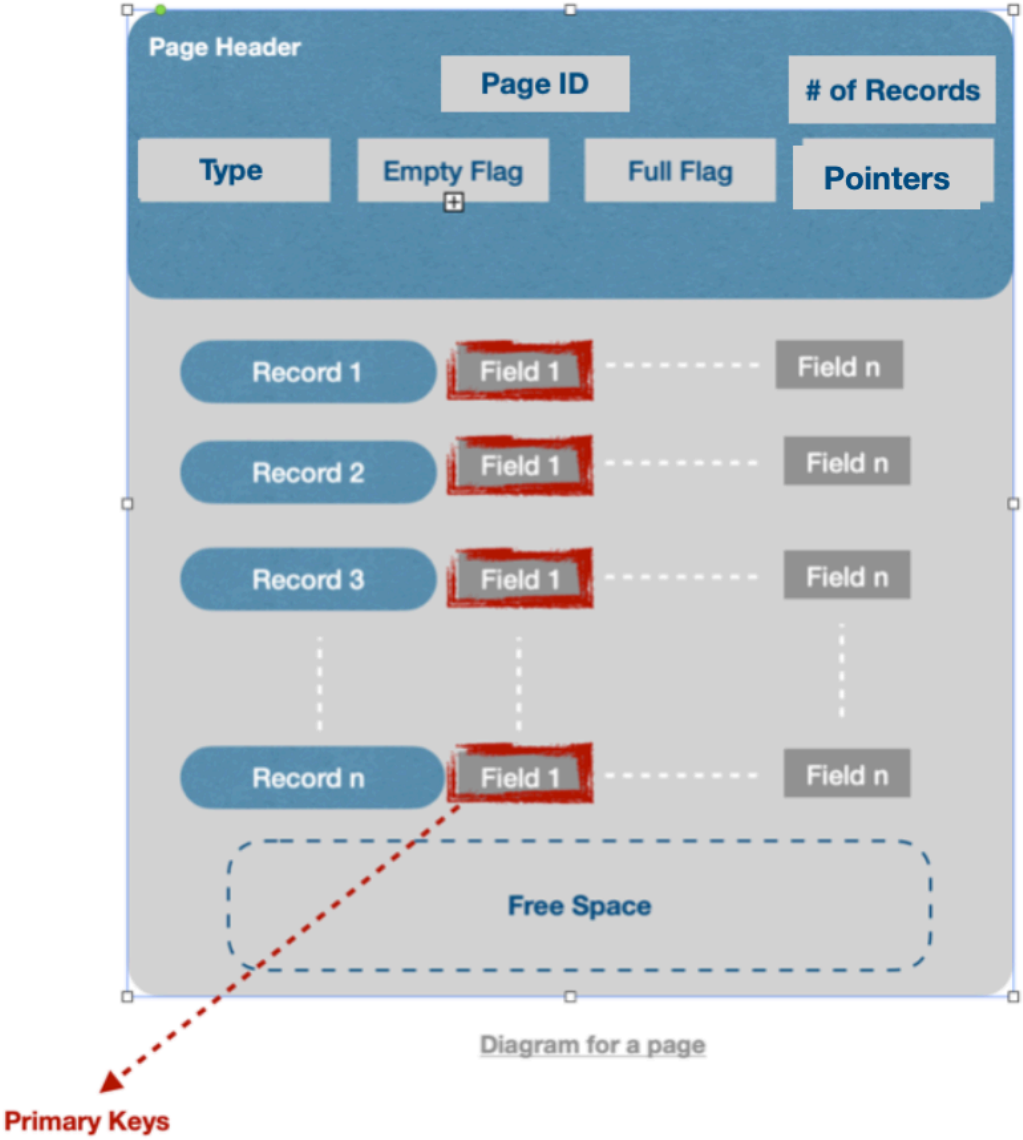

Diagram for a page

3.4 RECORD

Records are the data storage units that appears at rows in my system and a record is composed of 3 fields minimum and 10 fields at max so that reasonable amount of data is acceptable for one record.

If user enters more than 3 fields for a record then the new record is asked to be created by the system.

First field in a record is the primary key of that record.

Record Header keeps the following informations about that record:

- Record ID
- Empty Flag(1 if it is empty, 0 otherwise)
- Full Flag(1 if it is full, 0 otherwise)
- Record Type
- Number of Fields currently stored at corresponding record
- Id of the page that the record is currently exists

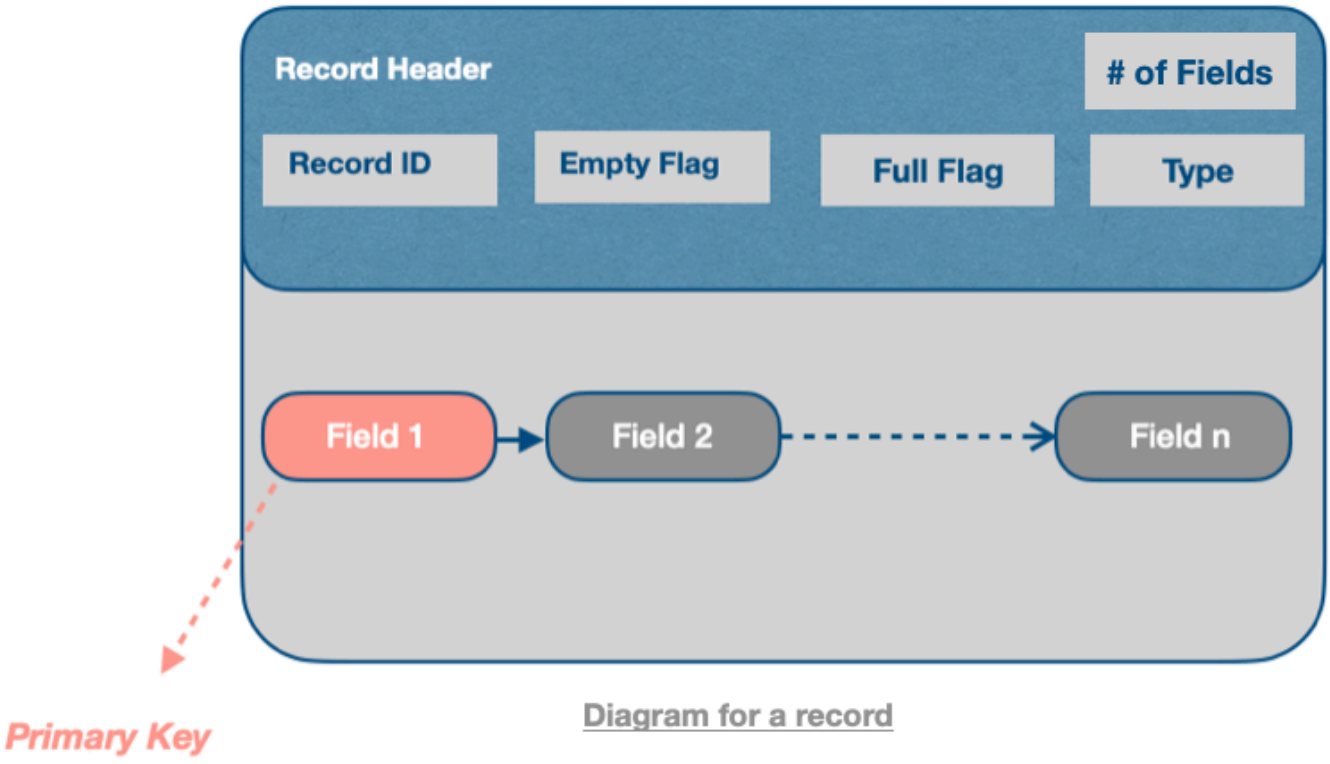

Diagram for a record

3.5 FIELD

Fields are the cells for database management system that are the smallest unit of informations that one can access.

In my database fields are **optional fields** as you may leave blank a field instead of necessarily enter a data.

Also my fields are **numeric fields** as the data stored in a field is consists of integers(decimals and the negative numbers also accepted).

Where as the name of a field shall be alphanumeric and can take max 24 and minimum 8 alphabetic characters and the **first field of a record is always the primary key.**

Diagram for a field

OPERATIONS

DDL operations

//Following algorithm will allow the user to create a type

createType() {

-All operations done in system catalog except for create new file for that type.

When new file will be created file manager take action and creates that file following the informations given in algorithm part.

```
INPUT -> typeName
INPUT -> (int)numberOfFields
create Type Struct newType-> type(typeName, numberOfFields)
open -> System Catalog
        open -> Data Types(of System catalog)
        add newType to Data Types.
        open ->File1
                //for next file pointers from file list
                while (file.next(linkedList) does not point NULL)
                        open FileLast //Opens Next File
                        pointerCheck=pointer for next File of File_n ;

         createFile(NewFile, FileId, newType);
         //createFile(File name, File id, file type)
         *FileLast_nextPointer=NewFile; //from file Linked list
         *NewFile_nextPoint=NULL;  //from file Linked list
         *NewFile_prevPoint=FileLast; //from file Linked list
      }
```

//Following algorithm will allow user to delete a type

//while deleting type all the datas stored as that type will also be deleted

deleteType() {

***Open system catalog and from the files of system catalog; traverse the files .While traversing the files with Storage Manager in each file file manager retrieves the file types until the requested type is found. Once intended file is found file manager operates deletion process for that file (Also when file is being deleted pages of that file will be deleted by giving addresses to disk manager and then they will be deleted from hard disk.)**

```
INPUT -> typeName
open -> System Catalog
        open -> Data Types(of System catalog)
        remove typeName from Data Types. //from dataTypes stored at system catalog
        open ->File1
                string typeFound="";
                typeFound=File1.type() //retrieves first file's type
                while ( ! (typeFound.equals(typeName ) )
                        open FileLast -> *file.nextPointer//Opens Next File
                        typeFound=FileLast.type ;
                filePrev=*FileLast.PrevPointer; //to change its next pointer
                retrieve file filePrev;
                fileNext=*FileLast.NextPointer; //to change its previous pointer in file
                                                                linkedlist
                retrieve file fileNext;
                if (fileNext is NULL) //if type stored at last file
                        { filePrev.nextPointer==NULL}
                else if (filePrev is NULL) //if type stored at first file
                        {fileNext.prevPointer==NULL}
```

```
                else { fileNext.prevPointer = filePrev;
                        filePrev.nextPointer=fileNext;
                        }
from System Catalog;
        -> # of files, # of pages, #of records will be updated accordingly
 delete FileLast completely  //if it has leaf files delete recursively

}
```

//Following algorithm will allow user to see all types listed

listAllTypes()

```
{
*From System Catalog The Types stored will be accessed and listed to the console
for user
open -> System Catalog
        int n=SystemCatalog.#ofTypes;
        Table[]; //to record types
        open -> File1(of System catalog) //retrieve first file recorded
        pointer_for_next_file=File1.nextPointer; //from file linked list
        Table.add -> File1.Type;
        for (int i=1 to n-1)
                { FileHold= open -> *pointer_for_next_file; //retrieve next file
                  pointer_for_next_file=FileHold.nextPointer;
                  Table.add->FileHold.Type; }
        print console -> Table
}
```

DML OPERATIONS

//Following algorithm will allow user to create a record

- **Once user gives the informations about the record, the system opens the system catalog and find the given type among types listing in the system catalog.**

- **Once intended type is found, my storage system request the file manager to get the address of file in the given type and the file system sends request the disk manager to retrieve the file and then file manager request the disk manager to get pages in that file by using pointers among pages.**
- **Then the disk manager pass pages to file manager and file manager pass pages to my storage system.**
- **My system checks the pages to find enough space in the pages if empty space is found, record is pushed into the page, if not, my system request file manager to retrieve next page and the file manager reads the address of page and gives it to the disk manager to retrieve next pages.**
- **This process continues until proper page is found, if not, new page is created.**

```
createRecord() {
INPUT -> recordType;
INPUT -> recordID;
int numberOfFields=Type.fieldNumber;
open -> System Catalog
        SystemCatalog.#ofRecords++;
        open -> File1(of System catalog) //retrieve first file recorded
        pointer_for_next_file=(linkedList files)File1.nextPointer;
        currentType=File1.type;
        FileHold;
        while(currentType != recordType) {
                FileHold= open -> *pointer_for_next_file; //retrieve next file
                pointer_for_next_file=FileHold.nextPointer;
               currentType = FileHold.type;
        }
        open ->FileHold; //goes to the file that has same type with record;
        //If file has leaf files -> go until the bottom and open it
        load -> FileHold.FirstPage; //retrieves the first page from that file
        boolean isFull=1;
        while(isFull ==Page.Header.FullFlag) {
                If (Page.Header.nextPointer == NULL)
                        { createNewPage(); //this function creates page with intended
                                           //qualifications and checks if the file is full or
```

```
                                        //not. If it is full creates files required and
                                        //creates the page in that very file
                        open -> Page; //which is just created
                        break;}
                open ->*Page.Header.nextPointer; //go to next page;
        }
      open Page.lastRecord //by traversing record starting from first record

  RecordNew = createRecord(recordType,recordID);
  lastRecord.Header.nextPointer = RecordNew;
  RecordNew.Header.prevPointer = lastRecord;
  RecordNew.Header.nextPointer = NULL;
  for(i=1 to i<=numberOfFields)
        {       INPUT ->fieldName;
                INPUT ->NumericData;
                FieldNew=create Field(fieldName,NumericData);
                RecordNew.fields.add->FieldNew;
    }
}
```

//following algorithm will be used to delete record;

deleteRecord(){

- **User gives the informations about the record to be deleted(Record id and type),**
- **The system opens the system catalog and find the given type among its files' types**
- **Once intended file is found, my storage system request the file manager to get the addres of file in the given type and the file system sends request**

the disk manager to retrieve the file and then file manager request the disk manager to get pages in that file by using pointers among pages.

- **Then the disk manager pass pages to file manager .**
- **File manager checks one by one the pages retrieved from disk manager to find intended record to be deleted by looking their record header part.**
- **Storage manager retrieves the page from file manager to delete intended record and record is pulled out of the page and deleted.**

```
INPUT -> RecordId;
Input->RecordType;
open -> System Catalog
                open ->File1
                 string typeFound="";
                 typeFound=File1.type() //retrieves first file's type
                 while ( ! (typeFound.equals(RecordType ) )
                                  open FileLast -> *file.nextPointer//Opens Next File
                                  typeFound=FileLast.type ;
                 open->FileLast;
                 load->firstPage from FileLast;
                 id_hold=firstPage.firstRecord.RecordHeader.ID;
          while( id_hold != RecordId ){
                 currentRecord.Header.nextPointer; //retrieves next record
                 If(currentRecord.Header.nextPointer is NULL)//last record from page
                          {retrieve->next page; //by using currentPage's next pointer
                           currentRecord=nextPage.FirstRecord; }
         }
          prevRecord=currentRecord.Header.prevPointer;
          nextRecord=currentRecord.Header.nextPointer;
          prevRecord.Header.nextPointer=nextRecord;
          nextRecord.Header.prevPointer=prevRecord;
         delete -> Current Record from System
}
```

```
  //following function searches for a record by primary key
searchByPrimaryKey() {
INPUT->FieldName (of primary Key);
INPUT->NumericData (of primary Key);
open -> System Catalog;
        open ->First File;
                load -> FirstPage of File as Current page;
        currentRecord=currentPage.FirstRecord;
        keyHold=currentRecord.PrimaryKey.FieldName;
        numericHold=currentRecord.PrimaryKey.NumericData;
        while( (keyHold !=FieldName) && (numericHold !=NumericData)) {
                if(currentRecord.Header.nextPointer is Null) {//if last record of page
                        load (new)currentPage -> as currentPage.Header.NextPointer;
                        currentRecord=currentPage.firstRecord;
                        keyHold=currentRecord.PrimaryKey.FieldName;
                        numericHold=currentRecord.primaryKey.NumericData;
                        continue; }

                 else if( currentPage  is the last page of that file)
                        { open-> (new)currentFile //as next File
                        load (new)currentPage //currentFile.firstPage
                        currentRecord -> firstRecord of the page
                        keyHold=currentRecord.primaryKey.FieldName;
                        numericHold=currentRecord.primaryKey.NumericData;
                        continue; }
                 else {
                        recordHold=currentRecord.Header.NextPointer;
                        currentRecord=recordHold;
                        keyHold=currentRecord.primaryKey.FieldName;
                        numericHold=currentRecord.primaryKey.NumericData;  }
        }
```

//Final CURRENTRECORD is the record found by Primary KEY

}

//Following algorithm lists all records for a type

***Storage manager looks among its Files' types to see if it has intended type to be listed.**

Once the intended file is found ;

Storage manager request the file manager to get the address of file requested

The file system sends request the disk manager to retrieve the file

Then file manager request the disk manager to get pages in that file by using pointers among pages.

- **Then the disk manager retrieves and passes pages to file manager one by one**
- **file manager pass pages to my storage manager.**
- **storage manager gives the all datas for a record from the records of a page that is currently retrieved**

list_all_for_type() {

```
INPUT -> typeNAME;
open -> System Catalog;
        open ->First File;
        pointer_for_next_file=File1.nextPointer;
        currentType=File1.type;
        FileHold;
        while(currentType != typeName) {
                FileHold= open -> *pointer_for_next_file; //retrieve next file
                pointer_for_next_file=FileHold.nextPointer;
                currentType = FileHold.type;
        }
```

```
        open ->FileHold; //goes to the file that has same type as asked;
         //If file has leaf files -> go until the bottom and open it
         print console each datas of the pages under the file
}
```

Conclusions & Assessment

Throughout the project I tried to design my system as accurate and efficient as I can.

The thing that confuses me most through the whole project is fixed field size for a type.

I fixed the field numbers for a type and so whether or not the user fills whole fields of that record the storage for these fields will be allocated for that record anyway. However I think it was efficient in terms of algorithm efficiency.