CMPE 321
Summer 2015

Yusuf Hakan Kalayci

Implementation of DBMS

Table Of Content

1.Introduction

1.1 Definition

This project is about the implementation of a simple storage manager for a DBMS. In this project this topic is divided into four categories and all of them are prepared separately. These are:

- Introduction
- Data structures
- Algorithms
- Conclusion and Evaluation

1.2 Abstract

The brief summary of the storage manager will be given here. The operations which must be included by the storage manager are listed below.

File Operations

- Create a file with specified record type
- Delete a record and its file
- Display all record types or file types

Record Operations

- Insert a record to a file
- Retrieve a record according to its key field.
- Delete a record according to its key field.
- Display all the records in a file

The database is representing with one folder and in root folder there are a folder and a file. This file represents System Catalog and it is named as sys.cat. On the other hand folder named as files contains filename.dat files for all files. The sys.cat file contains general information about files in database. The filename.dat files are includes all necessary information.

Assumptions

- The files are stored in binary format.
- All file and field names can be at most 20 characters.
- There can be at most 10 fields in a record.
- All fields of records are integer type.
- The page size is assumed to be 4096 bytes
- There must be only one key and a type cannot be created without a key.
- First field of each type is also key field.
- Key field cannot be duplicate value.
- Cannot be created empty type in other words every type must have key field.
- Type names must be unique.
- We assume that unhandled actions will not be tried.
    - For example
        - disallowed type names or field names.

- remove record or file which is not available

2. Data Structures

Our DBMS is specialized in two different part:

- System catalog : This part stores information about current state of DBMS.
- Data : This part stores the information about specified type.

2.1 System Catalog

System catalog file consists of pages and each page has a page header and records. Every record represents a type.

i) Page

The figure of page is given below. First row represents Page Header all other rows represents a record.

- maxNumberOfRecords : It is calculated at the beginning and it represents maximum number of records that can fit into this page. It cannot be changed after page is created unless it is removed.
- numberOfRecords : It stores current number of records storing in this page.

maxNumberOfRecords	numberOfRecords
Record1	
Record2	
Record3 ...	

ii) Record

The figure of record is given below. First two column represents record header and all others represents body.

- numberOfFields : It stores the number of fields for this record type.
- notEmpty : It stores a boolean and this boolean show that if record is notEmpty(true) or empty(false).
- typeName : I represents name of type.
- fieldNames : They represents name of each field for this record.

numberOfFields	notEmpty	typeName	fieldName 1	fieldName 2	….

2.2 Data

In data files each file consist of pages and each page of them consist of records.

i) Page

The page type is complete same with page structure in system catalog. The figure of page is given below. First row represents Page Header all other rows represents a record.

- maxNumberOfRecords : It is calculated at the beginning and it represents maximum number of records that can fit into this page. It cannot be changed after page is created unless it is removed.
- numberOfRecords : It stores current number of records storing in this page.

maxNumberOfRecords	numberOfRecords
Record1	
Record2	
Record3 ...	

ii) Record

The figure of record is given below. First two column represents record header and all others represents body.

- notEmpty : It stores a boolean and this boolean show that if record is notEmpty(true) or empty(false).
- fields : They represents value of each field for this record and all of them are integer variable.

notEmpty	field 1	field 2	field 3	….

3 Code of Program

3.1 General Operations

Classes with related this topic are stored in **main** package and they are builded for executing begining operations.

i) Starter class

In this class we build our starting part of project. At the begining of application our program starts from this main function then runs Starter object. After starting, depending on users input program executes some DDL operations or DML operations.

```
package main;

import java.util.ArrayList;
import java.util.Scanner;
import syscat.DDL;
import data.DML;

public class Starter {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Starter st = new Starter();
		st.run();
	}

	public void run() {

		Scanner in = new Scanner(System. in );

		while (true) {
			System.out.println("================================================");
			System.out.println("Main Menu");
			System.out.println("0 : Exit");
			System.out.println("1 : DDL Operations");
			System.out.println("2 : DML Operations");
			System.out.println("================================================");
			int tmp = in .nextInt();
			if (tmp == 0) break;
			if (tmp == 1) runDDL();
			if (tmp == 2) runDML();
		}
	}

	public void runDDL() {
		DDL ddl = new DDL();
		ddl.run();
	}
	public void runDML() {

		Scanner in ;
		DML dml;
		DDL ddl = new DDL();

		in = new Scanner(System. in );

		while (true) {

			System.out.println("----------------------------------------");
```

```
                        System.out.println("DML Menu");
                        System.out.println("0 : exit");
                        System.out.println("1 : Add a new record to a file");
                        System.out.println("2 : Retrieve a record from a file");
                        System.out.println("3 : Delete a record from a file");
                        System.out.println("4 : Retrieve all records from a file");
                        System.out.println("-----------------------------------------------");

                        int tmp = new Integer( in .next()).intValue();

                        if (tmp == 0) break;
                        System.out.print("Please write file(type) name:");
                        String typeName = in .next();
                        syscat.Record type = ddl.findType(typeName);

                        dml = new DML(type, in );

                        if (tmp == 1) {
                                boolean status = dml.addRecord();
                                if (status) System.out.println("STATUS OK!");
                                else System.out.println("STATUS NOT OK!");
                        }
                        if (tmp == 2) {
                                data.Record rec = dml.retrieveRecord();
                                if (rec != null) System.out.println("    " + rec.toString(type));
                                else System.out.println("STATUS NOT OK!");
                        }
                        if (tmp == 3) {
                                boolean status = dml.deleteRecord();
                                if (status) System.out.println("STATUS OK!");
                                else System.out.println("STATUS NOT OK!");
                        }
                        if (tmp == 4) {
                                ArrayList < data.Record > res = dml.retrieveAllRecords();
                                for (int i = 0; i < res.size(); i++)
                                System.out.println("           " + res.get(i).toString(type));
                        }
                }
        }
}
```

3.2 Additional Operations

Classes with related this topic are stored in **extra** package and they are builded for executing begining operations.

i) Converter Class

This class is builded for converting variables from byte to required type or vice versa. It is builded to make simple to other classes.

```
package extra;

public class Converter {

	public static int byteToInt( byte ar[],int pos ){
		int res=0;
		for( int i=0;i<4;i++ )
		{
			res<<=8;
			res+=(int)ar[pos+i];
		}
		return res;
	}

	public static byte[] intToByte( int k ){
		int m=1<<8;
		byte res[]=new byte[4];
		for( int i=3;i>=0;i-- ){
			res[i]=(byte)(k%m);
			k>>=8;
		}
		return res;
	}

	public static String byteToString( byte ar[],int pos,int len ){
		String res="";
		for( int i=0;i<len;i++ )
			if( ar[pos+i]!=0 )
				res+=(char)ar[pos+i];
		return res;
	}

	public static byte[] stringToByte( String str,int len ){
		byte res[]=new byte[len];
		for(int i=0;i<len;i++)
			res[i]=0;
		for(int i=0;i<str.length();i++)
			res[i]=(byte)str.charAt(i);
		return res;
	}

	public static boolean byteToBool( byte ar[],int pos ){
		if( ar[pos]==(byte)0 )
			return false;
		return true;
	}

	public static byte[] boolToByte( boolean f ){
		byte res[]=new byte[1];
		if(!f)
			res[0]=(byte)0;
```

```
                else
                        res[0]=(byte)1;
                return res;
        }

}
```

3.3 DDL Operations

Classes with related this topic are stored in **syscat** package and they are builded for executing DDL operations.

i) DDL Class

This class is builded for executing basic DDL operations. All functions are named with names which are related their purposes. These functions accomplish their purposes while using other classes under this package.

```
package syscat;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Scanner;

public class DDL {

        public final int PAGE_SIZE = 4096;

        public void run(){

                Scanner in = new Scanner(System.in);

                while(true){

                        System.out.println("-----------------------------------------");
                        System.out.println("DDL menu:");
                        System.out.println("0 : Exit");
                        System.out.println("1 : Add a new file type");
                        System.out.println("2 : Delete a file type");
                        System.out.println("3 : Display all file types");
                        System.out.println("-----------------------------------------");
                        int r = in.nextInt();

                        if(r==0)
```

```
                        break;
                if(r==1){
                        System.out.println(this.createType());
                }
                if(r==2)
                {
                        System.out.println(this.deleteType());
                }
                if(r==3)
                {
                        System.out.println(this.displayAllTypes());
                }

        }
}

public boolean createType(){

        System.out.println("Create Type");

        Scanner in = new Scanner(System.in);

        System.out.print("Please write type name: ");
        String typeName;
        typeName = in.next();

        System.out.print("Please write number of fields : ");
        int numberOfFields;
        numberOfFields = in.nextInt();

        String fieldNames[] = new String[numberOfFields];
        for(int i=0;i<numberOfFields;i++)
        {
                System.out.print("write name of field " + i + " : ");
                fieldNames[i] = in.next();
        }

        URL url = getClass().getResource("/sys.cat");

        createFile( typeName );

        byte ar[] = new byte[PAGE_SIZE];

        int position=0;

        Page page;

        RandomAccessFile rac=null;

        try {
                rac = new RandomAccessFile(new File(url.getPath()), "rw");
                while(true){
```

```
                                try{
                                        page = new Page(rac,position);
                                }catch(Exception e){
                                        page = new Page();
                                        page.writePage(rac,position);
                                }

                                if( page.isEmpty() ){
                                        page=new Page();
                                        page.writePage(rac,position);
                                }
                                if( !page.isFull() ){
                                        Record rec = new Record(typeName, numberOfFields, fieldNames);
                                        page.insertRecord(rec);
                                        page.writePage(rac,position);
                                        rac.close();
                                        return true;
                                }
                                position += PAGE_SIZE;
                        }

                } catch (FileNotFoundException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return false;

        }

        public ArrayList<Record> displayAllTypes(){

                System.out.println("Display All Types");

                URL url = getClass().getResource("/sys.cat");

                byte ar[] = new byte[PAGE_SIZE];

                int position=0;

                Page page=null;

                RandomAccessFile rac=null;

                ArrayList<Record> result = new ArrayList<Record>();

                try {
                        rac = new RandomAccessFile(new File(url.getPath()), "rw");
                        while(true){
```

```
                                try{
                                        page = new Page(rac,position);
                                }catch(Exception e){
                                        //e.printStackTrace();
                                        break;
                                }

                                if( !page.isEmpty() ){

                                        result.addAll( page.displayAllRecords() );

                                }
                                position += PAGE_SIZE;
                        }
                        rac.close();
                } catch (FileNotFoundException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return result;

        }

        public boolean deleteType(){

                System.out.println("Delete a Type");

                Scanner in = new Scanner(System.in);

                System.out.print("Please write type name to delete : ");
                String typeName;
                typeName = in.next();

                URL url = getClass().getResource("/sys.cat");

                byte ar[] = new byte[PAGE_SIZE];

                int position=0;

                Page page;

                RandomAccessFile rac=null;

                deleteFile(typeName);

                try {
                        rac = new RandomAccessFile(new File(url.getPath()), "rw");

                        while(true){
```

```
                                try{
                                        page = new Page(rac,position);
                                }catch(Exception e){
                                        break;
                                }
                                if( page.deleteRecord(typeName) ){
                                        page.writePage(rac, position);
                                        rac.close();
                                        return true;
                                }
                                position+=PAGE_SIZE;
                        }
                        rac.close();
                } catch (FileNotFoundException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return false;

        }

        public Record findType( String typeName ){

                URL url = getClass().getResource("/sys.cat");

                byte ar[] = new byte[PAGE_SIZE];

                int position=0;

                Page page;

                RandomAccessFile rac=null;

                try {
                        rac = new RandomAccessFile(new File(url.getPath()), "rw");

                        while(true){
```

```
                                try{
                                        page = new Page(rac,position);
                                }catch(Exception e){
                                        break;
                                }
                                Record rec = page.findRecord(typeName);
                                if( rec!=null ){
                                        rac.close();
                                        return rec;
                                }
                                position+=PAGE_SIZE;
                        }
                        rac.close();
                } catch (FileNotFoundException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return null;

        }

        private boolean createFile( String str ){

                URL url=null;
                try {
                        url = new URL(getClass().getResource("/files") + "/" + str + ".dat" );
                } catch (MalformedURLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                if(url==null)
                        return false;

                File file = new File(url.getPath());

                try {
                        file.createNewFile();
                } catch (IOException e) {
```

```
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return false;
                }

                return false;


        }

        private boolean deleteFile( String str ){

                URL url = getClass().getResource("/files/"+str+".dat");

                if(url==null)
                        return false;

                File file = new File(url.getPath());

                file.delete();

                return true;

        }

}
```

ii) Page Class

This class is builded for executing page operations on DDL. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package syscat;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;

public class Page {

        public final int PAGE_SIZE = 4096;

        PageHeader header;
        byte data[] = new byte[PAGE_SIZE];

        public Page(){
                for( int i=0;i<data.length;i++ )
                        data[i]=(byte)0;
                header = new PageHeader( Record.getSize() );
        }
```

```
	public Page( RandomAccessFile rac,int position ) throws IOException{

		rac.seek(position);
		rac.readFully(data, 0, PAGE_SIZE);
		header = new PageHeader( data,0 );

	}

	public boolean isEmpty(){
		if( header.maxNumberOfRecords==0 )
			return true;
		return false;
	}

	public boolean isFull(){
		if( header.maxNumberOfRecords>0 &&
header.numberOfRecords==header.maxNumberOfRecords )
			return true;
		return false;
	}

	public Record findRecord( String typeName ){

		int position = PageHeader.getSize();

		Record rec;

		while( PAGE_SIZE - position >= Record.getSize() ){

			rec = new Record(data, position);

			if( rec.header.notEmpty && rec.typeName.equals(typeName)){
				return rec;
			}

			position+=Record.getSize();

		}

		return null;

	}

	public boolean deleteRecord( String typeName ){

		int position = PageHeader.getSize();

		Record rec;

		while( PAGE_SIZE - position >= Record.getSize() ){

			rec = new Record(data, position);
```

```
                    if( rec.header.notEmpty && rec.typeName.equals(typeName)){

                            rec.header.notEmpty=false;
                            header.numberOfRecords--;
                            rec.writeRecord(data, position);
                            return true;
                    }

                    position+=Record.getSize();

            }

            return false;

    }

    public void insertRecord( Record record ){

            int position = PageHeader.getSize();

            Record rec;

            while( PAGE_SIZE - position >= Record.getSize() ){

                    rec = new Record(data, position);

                    if( rec.header.notEmpty==false ){
                            record.writeRecord(data, position);
                            header.numberOfRecords++;
                            break;
                    }

                    position+=Record.getSize();

            }

    }

    public ArrayList<Record> displayAllRecords(){

            ArrayList<Record> res = new ArrayList<Record>();

            int position = PageHeader.getSize();

            Record rec;

            while( PAGE_SIZE - position >= Record.getSize() ){

                    rec = new Record(data, position);

                    if( rec.header.notEmpty ){
                            res.add( rec );
```

```
                        }

                        position+=Record.getSize();

                }

                return res;
        }

        public void writePage( RandomAccessFile rac,int position ){

                byte tmp[] = header.headerInDbFormat();

                for(int i=0;i<tmp.length;i++)
                        data[i]=tmp[i];

                try {
                        rac.seek(position);
                        rac.write(data, 0, PAGE_SIZE);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                }

        }

}
```

iii) PageHeader Class

This class is builded for executing page header operations on DDL. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package syscat;

import extra.Converter;

public class PageHeader {

        public static int INT_SIZE = 4;
        public static int PAGE_SIZE = 4096;

        int maxNumberOfRecords;
        int numberOfRecords;

        public PageHeader( byte data[],int position ){
                maxNumberOfRecords = Converter.byteToInt(data, position);
                position+=INT_SIZE;
                numberOfRecords = Converter.byteToInt(data, position);
                position+=INT_SIZE;
        }
```

```
	public PageHeader( int recordSize ){
		maxNumberOfRecords = (PAGE_SIZE-2*INT_SIZE)/recordSize;
		numberOfRecords = 0;
	}

	public byte[] headerInDbFormat(){

		byte res[]=new byte[INT_SIZE*2];

		byte tmp[]=Converter.intToByte(maxNumberOfRecords);
		int pos=0;
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];
		tmp = Converter.intToByte(numberOfRecords);
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];

		return res;
	}

	public static int getSize(){
		return 2*INT_SIZE;
	}

}
```

iv) Record Class

This class is builded for executing record operations on DDL. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package syscat;

import extra.Converter;

public class Record {

	public RecordHeader header;
	public String typeName;
	public String fieldNames[];

	public Record( byte data[],int position ){

		this.fieldNames=new String[10];

		for( int i=0;i<10;i++ )
			this.fieldNames[i]="";

		header = new RecordHeader(data, position);
```

```
                position+=RecordHeader.getSize();

                typeName = Converter.byteToString(data, position, 20);
                position+=20;

                for( int i=0;i<header.numberOfFields;i++ )
                {
                        fieldNames[i] = Converter.byteToString(data, position, 10);
                        position+=10;
                }

        }

        public Record( String typeName,int numberOfFields,String fieldNames[] ){

                this.fieldNames=new String[10];

                for( int i=0;i<10;i++ )
                        this.fieldNames[i]="";

                this.header=new RecordHeader(numberOfFields);
                this.typeName=typeName;
                for( int i=0;i<numberOfFields;i++ )
                        this.fieldNames[i]=fieldNames[i];
        }

        public void writeRecord( byte data[],int position ){
                byte tmp[];
                tmp=header.headerInDbFormat();
                for( int i=0;i<tmp.length;i++ )
                        data[position++]=tmp[i];
                tmp=Converter.stringToByte(typeName, 20);
                for( int i=0;i<tmp.length;i++ )
                        data[position++]=tmp[i];
                for( int i=0;i<fieldNames.length;i++ )
                {
                        tmp=Converter.stringToByte(fieldNames[i], 10);
                        for(int j=0;j<tmp.length;j++)
                                data[position++]=tmp[j];
                }
        }

        @Override
        public String toString() {
                // TODO Auto-generated method stub
                return typeName;
        }

        public static int getSize(){
                return RecordHeader.getSize() + 20 + 10*10;
        }

}
```

v) RecordHeader Class

This class is builded for executing record header operations on DDL. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package syscat;

import java.text.spi.NumberFormatProvider;

import extra.Converter;

public class RecordHeader {
	public int numberOfFields;
	public boolean notEmpty;

	public static int INT_SIZE = 4;
	public static int BOOL_SIZE = 1;

	public static int getSize(){
		return INT_SIZE+BOOL_SIZE;
	}

	public RecordHeader( int numberOfFields ){
		this.numberOfFields = numberOfFields;
		notEmpty=true;
	}

	public RecordHeader( byte data[],int position ){
		numberOfFields = Converter.byteToInt(data, position);
		position+=INT_SIZE;
		notEmpty= Converter.byteToBool(data, position);
		position+=BOOL_SIZE;
	}

	public byte[] headerInDbFormat(){

		byte res[]=new byte[INT_SIZE+BOOL_SIZE];

		byte tmp[]=Converter.intToByte(numberOfFields);
		int pos=0;
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];
		tmp = Converter.boolToByte(notEmpty);
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];

		return res;
	}

}
```

3.4 DML Operations

Classes with related this topic are stored in **data** package and they are builded for executing DML operations.

i) DML Class

This class is builded for executing basic DML operations. All functions are named with names which are related their purposes. These functions accomplish their purposes while using other classes under this package.

```
package data;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.net.URL;
import java.util.ArrayList;
import java.util.Scanner;

public class DML {

	public final int PAGE_SIZE = 4096;

	syscat.Record type;
	URL url;
	Scanner in;

	public DML( syscat.Record type,Scanner in ){
		this.in = in;
		this.type=type;
		this.url = getClass().getResource("/data/"+type+".dat");
	}

	public boolean addRecord(){

		System.out.println("Create Record");

		int numberOfFields = type.header.numberOfFields;

		int fields[] = new int[ numberOfFields ];

		for(int i=0;i<numberOfFields;i++)
		{
			System.out.print( type.fieldNames[i]+": " );
			fields[i]=in.nextInt();
		}
```

```
		URL url = getClass().getResource("/files/"+type.typeName+".dat");

		int position=0;

		Page page;

		RandomAccessFile rac=null;

		try {
			rac = new RandomAccessFile(new File(url.getPath()), "rw");
			while(true){

				try{
					page = new Page(rac,position,numberOfFields);
				}catch(Exception e){
					page = new Page(numberOfFields);
					page.writePage(rac,position);
				}

				if( page.isEmpty() ){
					page=new Page(numberOfFields);
					page.writePage(rac,position);
					//rac.close();
				}
				if( !page.isFull() ){
					Record rec = new Record(fields, numberOfFields);
					page.insertRecord(rec);
					page.writePage(rac,position);
					rac.close();

					return true;
				}
				position += PAGE_SIZE;
			}
		} catch (FileNotFoundException e2) {
			// TODO Auto-generated catch block
			e2.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


		return false;


	}

	public boolean deleteRecord(){

		System.out.println("Delete a Record");

		System.out.print("Please write key of record to delete : ");
```

```
		int key;

		key = in.nextInt();

		URL url = getClass().getResource("/files/"+type.typeName+".dat");

		int position=0;

		Page page;

		RandomAccessFile rac=null;

		try {
			rac = new RandomAccessFile(new File(url.getPath()), "rw");

			while(true){

				try{

					page = new Page(rac,position,type.header.numberOfFields);

				}catch(Exception e){

					break;

				}

				if( page.deleteRecord(key) ){

					page.writePage(rac, position);

					rac.close();


					return true;

				}
				position+=PAGE_SIZE;

			}
			rac.close();
		} catch (FileNotFoundException e2) {
			// TODO Auto-generated catch block
			e2.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


		return false;
	}
```

```
public Record retrieveRecord(){

        System.out.println("Retrieve a Record");

        System.out.print("Please write key of record to retrieve : ");

        int key;

        key = in.nextInt();

        URL url = getClass().getResource("/files/"+type.typeName+".dat");

        int position=0;

        Page page;

        RandomAccessFile rac=null;

        try {
                rac = new RandomAccessFile(new File(url.getPath()), "rw");

                while(true){

                        try{

                                page = new Page(rac,position,type.header.numberOfFields);

                        }catch(Exception e){

                                break;

                        }

                        Record rec = page.findRecord(key);

                        if( rec!=null ){

                                rac.close();

                                return rec;

                        }
                        position+=PAGE_SIZE;

                }
                rac.close();
        } catch (FileNotFoundException e2) {
                // TODO Auto-generated catch block
                e2.printStackTrace();
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
                }

                return null;

        }

        public ArrayList<Record> retrieveAllRecords(){

                System.out.println("Retrieve All Records");

                URL url = getClass().getResource("/files/"+type.typeName+".dat");

                int position=0;

                Page page=null;

                RandomAccessFile rac=null;

                ArrayList<Record> result = new ArrayList<Record>();

                try {
                        rac = new RandomAccessFile(new File(url.getPath()), "rw");
                        while(true){

                                try{
                                        page = new Page(rac,position,type.header.numberOfFields);
                                }catch(Exception e){
                                        //e.printStackTrace();
                                        break;
                                }

                                if( !page.isEmpty() ){

                                        result.addAll( page.displayAllRecords() );

                                }
                                position += PAGE_SIZE;
                        }
                        rac.close();
                } catch (FileNotFoundException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return result;
        }
}
```

ii) Page Class

This class is builded for executing page operations on DML. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package data;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;

public class Page {

public final int PAGE_SIZE = 4096;

	PageHeader header;
	byte data[] = new byte[PAGE_SIZE];

	int numberOfFields;

	public Page( int numberOfFields ){
		this.numberOfFields = numberOfFields;
		for( int i=0;i<data.length;i++ )
			data[i]=(byte)0;
		header = new PageHeader( Record.getSize() );
	}

	public Page( RandomAccessFile rac,int position,int numberOfFields ) throws IOException{

		this.numberOfFields=numberOfFields;
		rac.seek(position);
		rac.readFully(data, 0, PAGE_SIZE);
		header = new PageHeader( data,0 );

	}

	public boolean isEmpty(){
		if( header.maxNumberOfRecords==0 )
			return true;
		return false;
	}

	public boolean isFull(){
		if( header.maxNumberOfRecords>0 &&
header.numberOfRecords==header.maxNumberOfRecords )
			return true;
		return false;
	}

	public Record findRecord( int key ){

		int position = PageHeader.getSize();
```

```
		Record rec;

		while( PAGE_SIZE - position >= Record.getSize() ){

			rec = new Record(data, position ,this.numberOfFields);

			if( rec.header.notEmpty && rec.fields[0] == key){
				return rec;
			}

			position+=Record.getSize();

		}

		return null;

	}

	public boolean deleteRecord( int key ){

		int position = PageHeader.getSize();

		Record rec;

		while( PAGE_SIZE - position >= Record.getSize() ){

			rec = new Record(data, position,this.numberOfFields);

			if( rec.header.notEmpty && rec.fields[0] == key){

				rec.header.notEmpty=false;
				header.numberOfRecords--;
				rec.writeRecord(data, position);
				return true;
			}

			position+=Record.getSize();

		}

		return false;

	}

	public void insertRecord( Record record ){

		int position = PageHeader.getSize();

		Record rec;

		while( PAGE_SIZE - position >= Record.getSize() ){

			rec = new Record(data, position ,this.numberOfFields);
```

```
                    if( rec.header.notEmpty==false ){
                        record.writeRecord(data, position);
                        header.numberOfRecords++;
                        break;
                    }

                    position+=Record.getSize();

                }

        }

        public ArrayList<Record> displayAllRecords(){

                ArrayList<Record> res = new ArrayList<Record>();

                int position = PageHeader.getSize();

                Record rec;

                while( PAGE_SIZE - position >= Record.getSize() ){

                        rec = new Record(data, position,this.numberOfFields);

                        if( rec.header.notEmpty ){
                                res.add( rec );
                        }

                        position+=Record.getSize();

                }

                return res;
        }

        public void writePage( RandomAccessFile rac,int position ){

                byte tmp[] = header.headerInDbFormat();

                for(int i=0;i<tmp.length;i++)
                        data[i]=tmp[i];

                try {
                        rac.seek(position);
                        rac.write(data, 0, PAGE_SIZE);
                } catch (IOException e) {
                        // TODO Auto-generated catch block

                        //e.printStackTrace();
                }

        }
```

```
}
```

iii) PageHeader Class

This class is builded for executing page header operations on DML. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package data;

import extra.Converter;

public class PageHeader {

	public static int INT_SIZE = 4;
	public static int PAGE_SIZE = 4096;

	int maxNumberOfRecords;
	int numberOfRecords;

	public PageHeader( byte data[],int position ){
		maxNumberOfRecords = Converter.byteToInt(data, position);
		position+=INT_SIZE;
		numberOfRecords = Converter.byteToInt(data, position);
		position+=INT_SIZE;
	}

	public PageHeader( int recordSize ){
		maxNumberOfRecords = (PAGE_SIZE-2*INT_SIZE)/recordSize;
		numberOfRecords = 0;
	}

	public byte[] headerInDbFormat(){

		byte res[]=new byte[INT_SIZE*2];

		byte tmp[]=Converter.intToByte(maxNumberOfRecords);
		int pos=0;
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];
		tmp = Converter.intToByte(numberOfRecords);
		for( int i=0;i<tmp.length;i++ )
			res[pos++]=tmp[i];

		return res;
	}

	public static int getSize(){
		return 2*INT_SIZE;
	}
```

```
}
```

iv) Record Class

This class is builded for executing record operations on DML. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package data;

import extra.Converter;

public class Record {

	public RecordHeader header;

	public int fields[];

	public static int INT_SIZE = 4;

	public Record( byte data[],int position,int numberOfFields ){

		this.fields=new int[10];

		for( int i=0;i<10;i++ )
			this.fields[i]=0;

		header = new RecordHeader(data, position);
		position+=RecordHeader.getSize();

		for( int i=0;i<numberOfFields;i++ )
		{
			fields[i] = Converter.byteToInt(data, position);
			position+=INT_SIZE;
		}

	}

	public Record( int fields[],int numberOfFields ){

		this.fields=new int[10];

		for( int i=0;i<10;i++ )
			this.fields[i]=0;

		this.header=new RecordHeader();

		for( int i=0;i<numberOfFields;i++ )
			this.fields[i]=fields[i];
	}
```

```
	public void writeRecord( byte data[],int position ){
		byte tmp[];
		tmp=header.headerInDbFormat();
		for( int i=0;i<tmp.length;i++ )
			data[position++]=tmp[i];
		for( int i=0;i<fields.length;i++ )
		{
			tmp=Converter.intToByte(fields[i]);
			for(int j=0;j<tmp.length;j++)
				data[position++]=tmp[j];
		}
	}

	public String toString( syscat.Record type ){

		String res = "";

		for(int i=0;i<type.header.numberOfFields;i++)
		{
			res+=type.fieldNames[i]+":"+ fields[i];
			if( i!=type.header.numberOfFields-1 )
				res += ",  ";
		}

		return res;

	}

	public static int getSize(){
		return RecordHeader.getSize() + 4*10;
	}

}
```

v) RecordHeader Class

This class is builded for executing record header operations on DML. On the other hand this class and its structure are appropriate for data structure mentioned in Data Structures part.

```
package data;

import extra.Converter;

public class RecordHeader {

	public boolean notEmpty;

	public static int BOOL_SIZE = 1;
```

```
        public static int getSize(){
                return BOOL_SIZE;
        }

        public RecordHeader(){
                notEmpty=true;
        }

        public RecordHeader( byte data[],int position ){
                notEmpty= Converter.byteToBool(data, position);
                position+=BOOL_SIZE;
        }

        public byte[] headerInDbFormat(){

                byte res[]=new byte[BOOL_SIZE];
                byte tmp[];

                int pos=0;
                tmp = Converter.boolToByte(notEmpty);
                for( int i=0;i<tmp.length;i++ )
                        res[pos++]=tmp[i];

                return res;
        }

}
```

4. Conclusion and Evaluation

To sum up, this project helped the understanding to logic of database management system and disk usage. While preparing project many problems are occurred and while solving this problems logic of database management system has been understood.

Although the project has done and java codes are implemented, there may be mistakes in logical aspects. In whole project limits and error handlings can cause some problems because they aren't handled. They should also be fixed.

On the other hand, we create a simple storage manager but it didn't developed considering maximizing performance. Therefore this issue is also a big topic then it should be examined.

In all operations and structures, I have used simple array data structure it is easy to think and decide. However, array data structure is very stable so it sometimes requires moving data from somewhere to another. These movings can cause time consuming and RAM usage problems. Therefore a more effective data structure can be used for this purpose.