CMPE321, 2018-2019-2, Database Storage Manager System, Ömer Faruk Özdemir 2016400048

Ömer Faruk Özdemir

April 6, 2019

Contents

1 Section One

1.1 Introduction

In this project I need to design a simple database storage manager system. I need to decide how to hold the records, design system catalogue, pages and records. I also need to create algorithms for DDL and DML operations.

Since this is a simple storage manager, I will not be doing error checking, like wrong input or duplicate records. However, if it is wanted they are easy to design or implement, since the main idea is done.

1.2 Assumptions & Constraints

- There is no null value for any record.
- Every record is integer.
- There is no bad entry, so I will not be doing replicate or wrong input checks.
- Since this is a simple storage manager I will pick max number of field in a type as 6.
- Maximum name length of a type or field name is 8.
- I will hold a specific type's records in same pages and same files.
- Each record is a line, meaning each record finishes with "\n".
- Each record is 64 bytes.
- Each page is 1KB at most.
- Each file is 8KB at most.
- Files will be txts and pages will be maximum of 16 record(lines) in a txt.
- When a page needs to be loaded to ram, only 1KB of file will be loaded, not whole txt.
- Type name, number of fields, field names, number of files, file names, number of lines in the file will be held in system catalogue.
- Every file will hold maximum of 8 pages. After that, a new file will be created. And file names will be held in the system catalogue.
- When a file becomes empty it will be deleted from disk, and its name will be deleted from system catalogue.
- File names will be like this for a type named Dog: Dog1, Dog2, Dog3, Dog4, Dog5...

1.3 Storage Structures

In theory, database takes necessary information from the system catalogue, for example a page's, file manager will request that page from the disk manager, and disk manager will retrieve the page from the disk and give it to file manager.

- System Catalogue:
  - There is no paging in system catalogue.
  - System catalogue will hold types line by line, meaning every type information finishes with "\n".
  - Each line is in this format, type's name, number of fields, fields' names, number of files, number of record(lines) for each file, names of files.

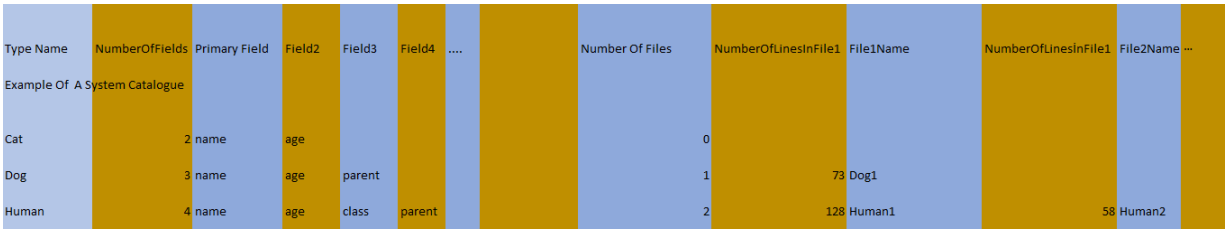

Figure 1: Picture of system catalogue.

- Files
  - A file will contain 8 pages at most.
  - A file will be 8KB at most.(1KBx8=8KB)
  - Files are in txt format, records are lines in that txt, and pages are blocks of 16 lines.
  - Files are made of pages. My max page number is 8 in a file. I will hold all records of a specific type in a file named of that type.
  - File names will be "typename"+"1", "typename"+"2", "typename"+"3"...
  - When every file is full, a new file will be created with increasing number of the last file's number.
  - When a file gets empty, meaning there is 0 records(line) in that file, and this can be accessed by the system catalogue, delete the file from both disk and its name from system catalogue.
- Pages

  - Pages are made of records.
  - My record limit in a page 16 records.
  - Since each record is a line, every page is 16 lines at most.
  - Each page is 1KB at most.(64byte*16=1KB)
  - There is no page header.

- Records
  - Each record is 1 line, meaning each record finishes with "\n".
  - Every field is a word, and a record is a line.
  - A record contains 6 fields at most.
  - Every field is an integer.
  - Every record is 64 bytes.
  - Fields are 8 char long at most, and we have 6 fields, 8byte*6=48 bytes. There are 7 space characters between fields, 1byte*7=7 bytes. "\n" makes 2 char, 1byte*2=2 bytes. 48+7+2=57 bytes. For number's sake, I increase every record to 64 bytes with space characters at the end.
  - There is no record header.

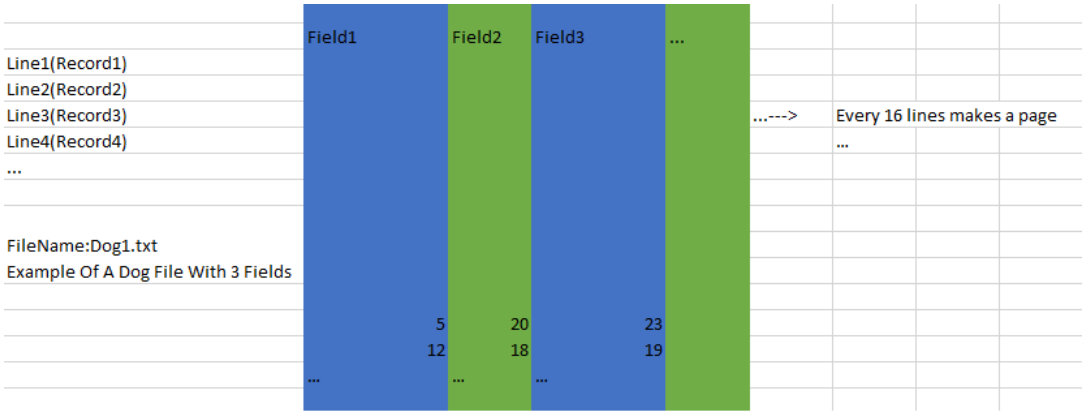

Figure 2: Picture of file

1.4 Operations

- DDL Operations
  - Create a type: Append type name and its field names to system catalogue with finishing 0. Since there are no records of that type, there is 0 file.

  - Delete a type: Reach to that type from system catalogue, get all file names about that type, delete all files about that type, then delete information related to that type from system catalogue. Note that there is no need to load pages or files to ram, just will delete files from the disk.
  - List all types: Read type names from system catalogue, sort them alphabetically, then print.

- DML Operations
  - Create a record: Find that types file names from the system catalogue, if there is no file name create a file with name "typename"+1 , else start from first file. If file is not full append the record to the last page, else continue with the next file.
    If all files are full create a new file, write that file name to system catalogue. Increase file number of that type by 1 in system catalogue, append the record to new file's first page.
    Increase number of records in the written file by 1.
  - Delete a record: Find that type's file names from the system catalogue, start from the first file, going page by page, scan primary values, find the correct record, delete the line of that record.
    Decrease record number of that file from system catalogue, if file is empty after deletion, delete the file from disk and delete its name from system catalogue.
    Decrease file number of that type by one from system catalogue.
  - Search for a record (by primary key): Find that type's file names from the system catalogue, start from the first file, going page by page, scan primary values, find the correct record, and get that line. Print it with field names from system catalogue.
  - List all records of a type: Reach to the system catalogue; find that type's file names, going page by page print all records.
  - Update a record: Search for that record by primary key, change the line of that record with update.

2 Algorithms

-Some global variables
-recordSize=64;
-recordNumInPage=16;
-pageNumInFile=8;

-I use bufferreader and bufferwriter with size of recordSize*recordNumInPage.
-This makes accessing files page by page automated.
–After that I read record line by line, and tokenize them and do some text process.

-Note: Accessing files page by page means just loading corresponding lines in that txt.
-getCommand() returns command part after 2 words(ie. part after create type)

-increaseRecordSizeToLimit: adds space to end of to string until it reaches to 62 length, 2 length comes from "\n"

Algorithm 1 Create a type

```
1: line=getCommand()
2: line+= " "+ 0
3: catWriter=new PrintWriter(new BufferWriter(new FileWriter ("systemcat-
   alogue.txt",true),recordSize*recordNumInPage))
4: catWriter.println(line)
```

//Get user input, which contains, typename fieldNumber field1name field2name... //Append it to system catalogue's txt le like this, typename fieldNumber field1name field2name ... 0

Algorithm 2 Delete a type

```
1: command=getCommand()
2: commandTokenizer=command.tokenize()
3: typeName=commandTokenizer.nextToken()
4: catReader=new BufferReader(new File("systemcatalogue.txt"),recordSize*recordNumInPage)

5: catWriter=new BufferWriter(new File("replicate.txt"),recordSize*recordNumInPage)

6: while (line=catReader.readLine())!=null do
7:    tokenizer=line.tokenize()
8:    if tokenizer.nextToken()==typeName then
9:       numberOfFields=ParseInt(tokenizer.nextToken())
10:      for int i=0;i<numberOfFields;i++ do
11:         tokenizer.nextToken()
12:      end for
13:      fileNumber=ParseInt(tokenizer.nextToken))
14:      for int i=0;i<fileNumber;i++ do
15:         tokenizer.nextToken()
16:         new File(tokenizer.nextToken()).delete()
17:      end for
18:   else
19:      catWriter.println(line)
20:   end if
21: end while
22: new File("systemcatalogue.txt").delete()
23: new File("replicate.txt").renameTo("systemcatalogue.txt")
```

//Search system catalogue line by line
// Reach to that type's line with the typeName
// Get fileNames from there.
// Delete every file.
// Delete the line of that type from system catalogue.

Algorithm 3 List all types

```
1: command=getCommand()
2: commandTokenizer=command.tokenize()
3: typeName=commandTokenizer.nextToken()
4: catReader=new BufferReader(new File("systemcatalogue.txt"),recordSize*recordNumInPage)
5: outputWriter=new BufferWriter(new File("output.txt"),recordSize*recordNumInPage)
6: while (line=catReader.readLine())!=null do
7:    tokenizer=line.tokenize()
8:    Name=tokenizer.nextToken
9:    outputWriter.println(line)
10: end while
```

// Search system catalogue line by line.
// Print typenames.

Algorithm 4 Create a record

```
1: command=getCommand()
2: typeName=command[typeName]
3: typeLine=""
4: catReader= BufferReader("systemcatalogue.txt) with size recordSize x
   recordNumInPage
5: catWriter= BufferWriter("systemcatalogue.txt) with size recordSize x
   recordNumInPage
6: typeLine=""
7: newTypeLine=""
8: for every line in catReader do
9:    if line[type]==typeName then
10:      typeLine=line
11:   else
12:      catWriter.println(line)
13:   end if
14: end for
15: bool writtenTheRecord=false
16: fieldNumber=typeLine[fieldNumber]
17: if fileNumber==0 then
18:   create File named typeName+"1"
19:   File(typeName+"1").append(increaseRecordSizeToLimit(command[field1,field2,...,field(fieldNumber]))

20:   writtenTheRecord=true
21:   newTypeLine= typeLine[type,fieldNumber,field1,..,field(fieldNumber)]+
      "1 1 " + typeName+"1"
22: else
      newTypeLine+= typeLine[typeName,fieldNumber,field1,...,field(fieldNumber]
      + " "
23: end if
24: for every fileName in typeLine do
25:   if typeLine[fileRecordNumber]¡ pageNumInFile x recordNumInPage
      then
26:      File(fileName).append(increaseRecordSizeToLimit(command[field1,field2,...,field(fieldNumber]))

27:      writtenTheRecord=true
28:      newTypeLine+= typeLine[fileRecordNumber]+1 " " + fileName
29:   else
30:      newTypeLine+= typeLine[fileRecordNumber] " " + fileName
31:   end if
32: end for
33: if !writtenTheRecord then
34:   newTypeLine=typeLine[type,fieldNumber,field1,...,field(fieldNumber,
      fileNumber+1, file1RecordNumber,File1....,file(fileNumber)RecordNumber,
      File(fileNumber)]
35:   newFileNum=ParseInt(File(fileNumber).changeString(typeName,""))
36:   newTypeLine+= " 1 " + typeName+newFileNum
37:   catWriter.println(newTypeLine)
38: end if
39: File("systemcatalogue").delete()
40: File("replicate").renameTo("systemCatalogue")
41: File(fileName+".txt").delete()
42: File("fileReplicate.txt").renameTo(fileName+".txt")
```

//Take input from the user. As “typename” “field1” “field2” ...
// Find typename’s line from the system catalogue (note that first word in every line is typeName).
// Get fileNumber.
//IFfileNumber==0
// Create a new txt file with name “typename”+1
// Increase fileNumber by 1 in the system catalogue.
//ENDIF
// Scan the fileNames and lineNumbers one by one
//IF- every file has recordSize x recordNumInPage line
// Create a new file with increasing number from lastfile’s number.
// Increase fileNumber by 1, write that file’s name to the end of system catalogue with lineNumber 1
//ENDIF
// Append the record to first file with line less than recordSize x recordNumInPage ’s last page as “field1” “field2” ..
// Increase that file’s lineNumber by 1 in the system catalogue.

Algorithm 5 Delete a record

```
1: command=getCommand()
2: typeName=command[type]
3: priKeyToBeUpdated=command[primaryValue]
4: readCat=BufferReader(File("systemcatalogue.txt"), recordSize x recordNumInPage)
5: writeCat=BufferWriter(File("replicate.txt"), recordSize x recordNumInPage)
6: newTypeLine=""
7: typeLine=""
8: for every line in readcat do
9:    if line[type]=typeName then
10:      typeLine=line
11:   else
12:      writeCat.println(line)
13:   end if
14: end for
15: deletedTheRecord=false
16: for every fileName in typeLine do
17:   readFile=BufferReader(File("fileName.txt"), recordSize x recordNumInPage)
18:   writeFile=BufferWriter(File("fileReplicate.txt"), recordSize x recordNumInPage)
19:   for every line in readFile do
20:      if line[field1Value]==priKeyToBeUpdated then
21:         updatedTheRecord=true
22:      else
23:         writeFile.println(line)
24:      end if
25:   end for
26:   if deletedTheRecord then
27:      if fileRecordNum==1 then
28:         File(fileName+".txt").delete()
29:         newTypeLine=typeLine[type,fieldNumber,[fields], fileNumber-1, [files]except fileName)
30:      else
31:         newTypeLine=typeLine[type,fieldNumber,[fields], fileNumber, [file1Line file1Name ... fileLine-1 fileName ... ]
32:      end if
33:      File (fileName+".txt").delete()
34:      File ("fileReplicate.txt").renameTo(fileName+".txt")
35:      break;
36:   end if
37:   writeCat.println(newTypeLine)
38: end for
39: File ("systemcatalogue.txt").delete()
40: File ("replicate.txt").renameTo("systemcatalogue.txt")
```

//Scan system catalogue
// Reach to the line of that type
// Get the file names one by one(since there is no error input, there should be at least 1 files of that type)
// Scan file page by page, scan pages line by line
//IF first word in the line == givenPrimaryValue
// Delete that line
// Decrease number of lines of that file from system catalogue by 1.
// ENDIF
//IF number of lines==0
// Delete the file from disk, delete file name and 0 from the system catalogue, decrease file number by 1 from system catalogue
// ENDIF

Algorithm 6 Search for a record (by primary key)

```
1: command=getCommand()
2: typeName=command[type]
3: searchedPrimaryKey=command[primaryValue]
4: readCat=BufferReader(File("systemcatalogue.txt"),recordSize x recordNumInPage)
5: typeLine=""
6: outputWriter=BufferWriter(File("output.txt"),recordSize x recordNumInPage)
7: for every line in readcat do
8:    if line[type]=typeName then
9:       typeLine=line[type]
10:   end if
11: end for
12: bool found=false
13: for every fileName in typeLine do
14:   fileReader=BufferReader(File(fileName),recordSize x recordNumInPage)

15:   for every line in fileReader do
16:      if line[primaryValue==searchedPrimaryKey] then
17:         outputWriter.println(line)
18:         found=true
19:         break;
20:      end if
21:   end for
22:   if found then
23:      break;
24:   end if
25: end for
```

// Scan system catalogue

// Reach to type's line
// Get type's file names.
// Access to files page by page, scan pages line by line.
// IF first word in the line == givenPrimaryValue
// Print the record the output.
// break;
// ENDIF

Algorithm 7 List all records of a type

```
1: command=getCommand()
2: typeName=command[type]
3: readCat=BufferReader(File("systemcatalogue.txt"),recordSize x recordNu-
   mInPage)
4: typeLine=""
5: outputWriter=BufferWriter(FileWriter("output.txt",true),recordSize x
   recordNumInPage)
6: for every line in readcat do
7:    if line[type]=typeName then
8:       typeLine=line[type]
9:    end if
10: end for
11: for every fileName in typeLine do
12:    fileReader=BufferReader(File(fileName),recordSize x recordNumInPage)

13:    for every line in fileReader do
14:       outputWriter.println(line)
15:    end for
16: end for
```

// Scan system catalogue.
// Get that type's file names.
// Access to files page by page, copy each line.
// Print every record one by one.

3 Conclusion & Assessment

- This was a simple storage manager. For a more advanced one, number limitations can be increased. Maximum field number, name lengths, type number can be increased.
- Fields just support integer. For a more advanced design other types can be supported as well.
- Since this is a simple design compression and efficiency are not real concerns. All records of a type are held in same files and pages. There is no dynamic relocation of pages or files. Records stay in the same place. For efficiency and compression some other configurations can be done.
- There is no security concern for this design. Nothing is encrypted, everything is public in txt files.
- This not an professional design, if it was created for usage in professional life it would need to be more professional. It is a simple design, and easy to understand.