CMPE 321 Summer 2019
Project2-Implementation of Storage Manager System

Ege Onur Güleç - 2015300084

July 29, 2019

1 Introduction

In this assignment I tried to implement the storage manager design for the first project. This report consists of the information of the details of the implementation. Also I made some minor changes in the design from previous project. This project is written in Java.

2 Assumptions and Constraints

1) User always creates valid input
2)Max number of fields are 10
3)Files consists of 7 pages.If more page is needed a new file is created
4) A page consists of 10 records
5) Max length of a type name or a field name is 10
6) System catalog includes type names number of fields they have.
7) All records of same type are stored in the same file like $\prec$ typeName $\succ$ $\prec$ numberofFile $\succ$ .txt e.g. cat0.txt cat1.txt etc
8) Search update and delete are made by primary key.

3 Structure Design

We begin by creating the system catalogue.

number of types					
first type	number of files	number of pages	number of records	first field	...
second type	number of files	number of pages	number of records	first field	...
.....	....	.....	.....	.....	.....

Above table shows the design of system catalogue. It stores the metadata about the system only difference from first project is the locations of files corresponding to a type name are stored in disc directory

3.1 Type

.TypeName 12 byte
.isdeleted 1 byte
.NumberofFields 4byte
.FieldName 12 byte
For simplicity I assumed there are only two types in one page.

3.2 File

A file has multiple pages in it. When all the pages in a file are full we can create a new a file and a page of the desired type and add, update or delete records from that new file until its all pages are also full or deleted. File is 4+7*(58+10*numberofFields*4) byte

numberofRecords (4byte)
page1
page2
page3
...
page7

Above table shows the structure of a file which can be created for any type. A file consists of 5 pages.

3.3 Page

A page consists of two different things. One of them is page header which includes an unique page id and the other is several records which is another structure I will explain in the next section. A page is consists of Page id 4 byte + numberofRecords is 4 byte also 10 record (a record is 5 byte + numberofFields*4 byte). Total is 58+10*numberofFields*4 byte

Page ID (4byte)
numberofRecord(4byte)

Above table shows the structure of the page header

Page Header
Record 1
Record 2
Record 3
...
Record 10

Above table shows the structure of a page which can be created for any type. A page consists of 10 records

3.4 Record

A record consists of two different things. First one is the record header which has the record id, info about whether a record is empty or not and a pointer to the next record. Second one is that a record has at most 10 fields store the relevant data for its specific type. Total a record is 5+numberofFields*4 byte.All fields are integer that is why a field is 4byte.

Record ID (4byte)
isdeleted (1byte)

Above table shows the structure of the record header

Record Header
First Field (4byte)
Second Field (4byte)
....
Tenth Field (4byte)

Above table shows the structure of a record

4 Algorithms for Operations

First DDL operations:

Input: typeName,number of fields,names of fields
systemCatalog.open();
numberofTypes=systemCatalog.readInteger() ;
numberofTypes++
while $i \leq numberofFields$ **do**
  dataType.fieldName[i]=namesofFields[i]
end
systemCatalogue.get(systemCatalogue.length());
systemCatalogue.write(typeName);
systemCatalogue.write(numberofFields);
systemCatalogue.write(false) (isdeleted) ;
while $i \leq numberofFields$ **do**
  systemCatalogue.write(dataType.fieldName[i] ;
end
systemCatalog.close();
$file \leftarrow createFile(typeName[0])$

Algorithm 1: Create a Type

Input: typeName
systemCatalog.open();
numberofTypes=systemCatalog.readInteger() ;
numberofTypes- -
currentpage=firstpage of systemCatalog ;
$k \leftarrow numberofFilesIntypeName$;
while $currentPage! = NULL$ **do**
    for *all types in page* **do**
        poppedTypeName=systemCatalogue.read();
        isdeleted=systemCatalogue.read();
        if *poppedTypeName==typeName isdeleted=false* **then**
            set isdeleted=true;
            systemCatalogue.write(isdeleted) ;
            return;
        else
            next type ;
        end
    end
    next page;
end
while $i \leq k$ **do**
    delete file named typeName[i].txt;
end
systemCatalog.close();

Algorithm 2: Delete a Type

while $currentPage! = NULL$ **do**
    for *all types in page* **do**
        poppedTypeName=systemCatalogue.read();
        isdeleted=systemCatalogue.read();
        if *isdeleted=false* **then**
            print(poppedTypeName)
        else
            next type ;
        end
    end
    next page;
end

Algorithm 3: List all Types

```
Input: typeName,recordID,recordFields
systemCatalog.open();
k ← typeName.numberofFiles in systemcatalog ;
while i ≤ k do
    typeName.file[i].open() ;
    if typeName.file[i].numberofRecords < 50 then
        for all pages p in typeName.file[i] do
            pageID=p.readInt() ;
            numberofRecords=p.readInt() ;
            if pageID == ithpage and numberofRecords < 10 then
                p.write(recordID);
                p.write(false) (isdeleted) ;
                while i ≤ recordFields.length() do
                    p.write(recordField) ;
                end
                typeName.file[i].numberofRecords++;
                return;
            else
                next page;
            end
        end
    else
        next file;
    end
end
construct a new file typeName[k+1];
set p.pageID=1 p.write(recordID);
p.write(false) (isdeleted) ;
while i ≤ recordFields.length() do
    p.write(recordField) ;
end
typeName.numberOfFiles++;
typeName.numberofRecords ++ ;
systemCatalog.close();
return r
```

Algorithm 4: Create a record

```
Input: typeName,recordID
systemCatalog.open();
k ← typeName.numberofFiles in systemcatalog ;
while i ≤ k do
    for all pages p in typeName.file[i] do
        if p.numberofrecords > 0 then
            for all records r in p do
                r.recordID=r.header.readByte[0](read first 4 bytes of header);
                if r.recordID==recordID then
                    isdeleted=r.read();
                    isdeleted=true;
                    r.write(isdeleted);
                    typeName.file[i].numberofRecords - -;
                    print("Record deleted successfully");
                    systemCatalog.close();
                    return
                else
                    next record;
                end
            end
        else
        end
        next page;
    end
end
print("Record does not exist");
systemCatalog.close();
return
```

Algorithm 5: Delete a Record

```
Input: typeName,recordID
systemCatalog.open();
k ← typeName.numberofFiles in systemcatalog ;
while i ≤ k do
    for all pages p in typeName.file[i] do
        for all records r in p do
            r.recordID=r.header.readByte[0](read first 4 bytes of header);
            r.isdeleted=r.header.readByte[4](read bytes 4 to 5);
            if r.recordID==recordID and r.isdeleted==false then
                print("Record is found");
                return r ;
            else
                next record;
            end
        end
    end
end
print("Record does not exist");
systemCatalog.close();
return
```

Algorithm 6: Search a Record (by primary key)

```
Input: typeName,recordID,recordFields
systemCatalog.open();
k ← numberofFilesIntypeName ;
while i ≤ k do
    for all pages p in typeName.file[i] do
        for all records r in p do
            r.recordID=r.header.readByte[0](read first 4 bytes of header);
            r.isdeleted=r.header.readByte[4](read bytes 4 to 5);
            if r.recordID==recordID and r.isdeleted==false then
                r.updateFields(recordFields) updating the given fields;
                print("Fields are updated");
                systemCatalog.close();
                return;
            else
                next record;
            end
        end
    end
end
print("Record does not exist");
systemCatalog.close();
return;
```

Algorithm 7: Update a Record(by primary key)

```
Input: typeName
systemCatalog.open();
k ← numberofFilesIntypeName ;
while i ≤ k do
    for all pages p in typeName.file[i] do
        for all records r in p do
            r.isdeleted=r.header.readByte[4](read byte 4 to 5 in header);
            if r.isdeleted==false then
                print(r);
            else
                next record ;
            end
        end
    end
end
systemCatalog.close();
```

Algorithm 8: List all Records

5 Changes from previous project

I resized the file, pages and records in comparison to previous project. I also do not use pointers since the project is written in java instead I use the locations of bytes to find next pages and records.

6 Conclusions and Assessment

In this project I implemented database management storage. My designs advantages are since I insert records linearly without any specific order insertion is fast however, as a disadvantage searching is costly since it looks page by page and record by record.
Another advantage is that I use page ids and record ids they make easier to find a page in a file or find a record in a page but as a disadvantage it allocates more memory. This ids make easier to implement my design.