DESIGN OF A SIMPLE STORAGE MANAGEMENT SYSTEM

ELIF ÇALIŞKAN

2016400183

CMPE321
SPRING 2019

Boğaziçi University

1 Introduction

This project is about designing a simple storage management system. A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data. The purpose of this project is creating a custom DBMS with the assumptions and decisions of our own. We are given some constraints and some operations -such as create type, delete type, list all types and create record, delete record, search for a record, list all records of a type- that our storage management system should support.

Assumptions and Constraints part contains some given constraints and my assumptions for the project. Storage Structures part contains some tables and graphics for representing the idea behind Operations part better. This part includes system catalog, file format, page format and record format along with their explanations.

2 Assumptions and Constraints

2.1 Assumptions

1. System Catalog
    - Format: sysCatalog.txt
    - There is no option for deleting the system catalog.
    - Type and field names can be minimum 1 maximum 10 bytes.
    - Type names must be unique.
    - Field and type names can be alphanumeric.
    - One type contains at least one field.
    - Every type must have a primary key which has an integer value.
    - For every type there will be read and write information. They define the permission for reading and writing. If a file is full: read=true, write=false; if a file is empty: read=false, write=true; if a file is deleted: read=false, write=false.
2. File
    - Format: <typeName>_file.txt
    - File size is 10KB.
    - File header includes: type name, pointer to first page, page count, write and read booleans

- Type and field names cannot exceed 10 characters.
- A type can at most have 10 fields.
- A file can contain multiple pages.
- A file can contain at most 10 pages.-> 1004bytes(page size)x 10 + 20 =10060 bytes < 10KB
- Each file holds one type of data.

3. Page
    - Page header includes: 4 bytes page id, a pointer to the following page, record count and read and write boolean flags.(In total: 14 bytes)
    - A page can contain at most 22 records.
    - Page size= 22x45(record size)+14 =1004 ~1024bytes = 1KB
4. Record
    - Max number of fields a type can have=10
    - Record header includes: 4 bytes page id, a pointer to the following record and valid boolean flags.

2.2 Constraints

1. Data must be organized in pages and pages must contain records.
2. Storing all pages in the same file is not allowed and a file must contain multiple pages.
3. The system must be able to create new files as storage manager grows.
4. When a file becomes free due to deletions, that file must be deleted.
5. Though a file contains multiple pages, the system must not load the whole file to RAM when it is needed. Instead, it must read a file page by page.

3 Storage Structures

3.1 System Catalog

System Catalog Header	PageId Pointer to next page number of types in a page

System Catalog file keeps the general information about types. This file is created with sysCatalog.txt name. It consists of pages since there can be numerous types in it. It has a header which holds the information of pageId, pointer to next page of System Catalog and number of types in a page. After the header: the type names, field count, field names and read, write flags are documented for each type. Type and field names can be alphanumerical and they can have at most 10 characters.

Type Name 1	Field Count 1	FieldName 1	FieldName 2	...	read	write
Type Name 2	Field Count 2	FieldName 1	FieldName 2	...	read	write
Type Name 3	Field Count 3	FieldName 1	FieldName 2	...	read	write
...	...	...	...	...	...	...

- Header: 12 bytes
  - pageId: 4 bytes (integer)
  - pointer to next page: 4 bytes
  - number of types: 4 bytes
- File: 116 bytes
  - type name: 10 bytes
  - field count : 4 bytes
  - field name: 10 bytes 10x10=100 bytes for 10 fields
  - read and write: 2x1=2 bytes

3.2 File

File Header	Type Name Pointer to First Page Page Count Write flag Read flag

File consists of pages. To keep track of pages, I created a header which includes type name, pointer to first page, page count, write and read flag. Every file has a limited size and page count has a maximum value of 10. Write and read flags keep the information of empty, full or deleted.

- Header: 20 bytes
  - type name: 10 bytes
  - pointer to first page: 4 bytes
  - page count: 4 bytes
  - read and write flag: 2x1=2 bytes

3.3 Page

Page Header	Page Id Pointer to Next Page Record Count Write flag Read flag

A page consists of page header and records. For the order of pages, I keep a pointer to the following page.

recordNumber	valid bit	Field Value 1	Field Value 2	...	Field Value 10
recordNumber	valid bit	Field Value 1	Field Value 2	...	Field Value 10
recordNumber	valid bit	Field Value 1	Field Value 2	...	Field Value 10
...	...	...	...	...	...

- Header: 14 bytes
  - pageId: 4 bytes
  - pointer to first page: 4 bytes
  - record count: 4 bytes
  - read and write flag: 2x1=2 bytes
- Record: 45 bytes
  - valid bit: 1 byte
  - record number: 4 bytes
  - field value: 4 bytes (4x10=40 bytes)

4 Operations

4.1 DDL Operations

4.1.1 Create a type

```
\\creates a new type and adds it to System Catalog
function createType(string typeName, string[] fieldNames)
\\controls if typeName is appropriate for size restriction and unique in System Catalog
if isProperTypeN(typeName) and isUniqueTypeN(typeName) then
  \\the file is created with the related header information
  file <- createFile(tableName_file.txt)
  file.setHeader(tableName, fieldNames, write:true, read:false)
  openFile(sysCatalog.txt)
```

```
  found <- false
  \\checks if there is any deleted type in System Catalog, if there is the
  new file is overwritten
  for all pages in System Catalog do
    for all types in pages do
      if type.read=false and type.read=false then
        type.setInformation(file)
        found <- true
      end if
    end for
  end for
  \\if there is no empty slot, then it adds the file to the end of System
  Catalog. If the pages of System Catalog are full the method also creates
  a page and links it to System Catalog
  if found=false then
    addToSysCatalog(typeName, file)
  end if
  closeFile(sysCatalog.txt)
else
  \\if type name is not suitable then error message is printed
  Print "error in type name"
end if
End function
```

4.1.2 Delete a type

```
\\deletes a type and updates System Catalog
function deleteType(string typeName)
openFile(sysCatalog.txt)
\\controls if typeName exists in System Catalog
if typeExistsInSysCatalog(typeName) then
  \\controls if file is already deleted by checking the System Catalog if
  read and write attributes are both false
  if isDeleted(typeName) then
    Print "<typeName> is already deleted."
  else
    \\every record in a file with the type of <typeName> should be deleted.
    For this, it is sufficient to invalidate every type in System Catalog with
    the same typeName.
    for all pages in System Catalog do
      for all types in page do
        if type.typeName=typeName then
          type.write <- false
          type.read <- false
```

```
            page.numberOfTypes–;
          end if
        end for
      end for
    else
      \\if typeName doesn't exist in System Catalog it prints an error mes-
      sage
      Print "specified type doesn't exist in System Catalog"
    end if
    closeFile(sysCatalog.txt)
    End function
```

4.1.3 List all types

```
\\lists all types in System Catalog
function listTypes
openFile(sysCatalog.txt)
\\to list all types in the system, it should go through every page in Sys-
tem Catalog and print every type which is not deleted
for all pages in System Catalog do
  for all types in page do
    if type.read=true or type.write=true then
      Print type.typeName
    end if
  end for
end for
closeFile(sysCatalog.txt)
End function
```

4.2 DML Operations

4.2.1 Create a record

```
\\creates a record and adds it to related page
function createRecord(string typeName, integer[] fieldValues)
found <- false
file <- getFile(typeName)
\\it goes over every page in a file and tries to find a page which is neither
full nor deleted
for all pages in file do
  \\checks if the page is full or deleted
  if page.write = true then
    \\if this record is the first record on this page then it makes read
    field true
    if page.write=true and page.read=false then
```

```
            page.read=true
        end if
        \\it goes over every record in a page and tries to find an empty spot
        for all records in page do
            if record.valid=false then
                record.valid <- true
                record.fields <- fieldValues
                found <- true
                page.recordCount++;
                break
            end if
        end for
        \\it checks if the added record makes the page full and updates
        write field accordingly
        if page.recordCount=22 then
            page.write <- false
        end if
    end if
    if found = true then
        break
    end if
  end for
  \\if there is no empty spot in pages, it creates a new page and points the
  last page in file to it
  if found = false and file.pageCount <10 then
    lastPage <- createPage(pageId, recordCount=0, write=true, read=false)
    file.addPage(lastPage)
    file.pageCount++
    createRecord(typeName, fieldValues)
  end if
else
  \\if the pageCount exceeds its limit a new file can be created
  createType(typeName(1))
  createRecord(typeName(1), fieldValues)
end if
End function
```

4.2.2 Delete a record

```
\\deletes a record and updates the related page
function deleteRecord(string typeName, integer primaryKey)
file <- getFile(typeName)
\\it searches every page in a file, if a page has read permission (not empty
or not deleted) it searches every record in that page
```

```
for pages in file do
  if page.read=true then
    for records in page do
      if record.primaryKey=primaryKey then
        record.valid <- false
        page.recordCount–;
        \\after deleting the record, it checks whether the page is empty
        if page.recordCount=0 then
          page.read <- false
          file.pageCount–
          \\if the page is empty, pageCount of file is decreased and if
          there is no page of that file, type is deleted from the System
          Catalog with deleteType method
          if file.pageCount=0 then
            deleteType(file.typeName)
          end if
        end if
        break
      end if
    end for
  end if
end for
End function
```

4.2.3 Search for a record (by primary key)

```
\\deletes a record and updates the related page
function searchRecord(string typeName, integer primaryKey)
file <- getFile(typeName)
\\it searches every page in a file, if a page has read permission (not empty
or not deleted) it searches every record in that page
for pages in file do
  if page.read=true then
    for records in page do
      if record.primaryKey=primaryKey and record.valid=true then
        return record
      end if
    end for
  end if
end for
\\if the wanted record does not exist it returns NULL
return NULL
End function
```

4.2.4 List of all records of a type

```
\\lists all records of a specified type
function listRecords(string typeName)
file <- getFile(typeName)
\\it goes over every page in a file, if a page has read permission (not empty
or not deleted) it iterates through every record in that page
for all pages in file do
  if page.read=true then
    for all records in page do
      if record.valid=true then
        Print record.fields
      end if
    end for
  end if
end for
End function
```

5 Conclusions and Assessment

This purpose of this project is to create a simple storage management design. For this, I used a System Catalog for keeping the general information like: type name, field names, number of fields, read and write flags. This System Catalog is also made of pages so the header holds the page id, pointer to next page and number of types in that page. System Catalog helps to find files and every file consists of pages. Every page has a header which keeps the information: pageId, pointer to next page, number of records in that page, read and write flags.

Addition to System Catalog is done by first finding a deleted type and then overwriting it if there is no deleted file then adding to the end of System Catalog. So creating a type might be relatively slow but it is memory efficient. Finding a type is improved with this method since the memory is not wasted and there are less pages to fetch which are used in System Catalog.

Creating a record is done with a similar manner to System Catalog. In order to use memory efficiently, the deleted records are overwritten first. But if there is no empty slot in the existing pages, it creates a page and links it to the end of the file. But if the file size is near the limit of 10KB, then a new file can be created with a slightly changed type name.

For pages, I used pointers to keep the order of a file. While searching a record, I need to go through every record in every page before the searched record. This can be improved by using a hash table.

In summary, this project has advantages and also disadvantages. Since I don't have an exact test input, I made the assumptions on my way of thinking, but the design can be improved with the consideration of real inputs.