

BOĞAZİÇİ UNIVERSITY

CMPE 321
DATABASE SYSTEMS

STORAGE MANAGER IMPLEMENTATION

PROJECT 2

BARAN DENİZ KORKMAZ
2015400183

SUMMER 19'

Contents

1	Introduction	2
2	Assumptions & Constraints	3
2.1	Assumptions	3
2.1.1	Base Assumptions	3
2.1.2	Extra Assumptions	4
2.1.3	Added Assumptions	4
2.2	Constraints	4
2.2.1	Base Constraints	4
2.2.2	Extra Constraints	4
2.2.3	Added Constraints	5
3	Storage Structures	5
3.1	System Catalogue	5
3.2	Data Storage Units	7
4	Operations	9
4.1	DDL Operations	9
4.1.1	Create a Type	9
4.1.2	Delete a Type	10
4.1.3	List All Types	10
4.2	DML Operations	11
4.2.1	Create a Record	13
4.2.2	Delete a Record	15
4.2.3	Search for a Record	16
4.2.4	Update a Record	17
4.2.5	List All Records of A Type	18
5	Conclusion & Assessment	19
5.1	Changes	19

1 Introduction

In the first project of the course, we are asked to design a storage manager which will be implemented later in the following projects. In this project, the storage manager implementation has been carried out. A storage manager is a system which is responsible from the definition and the manipulation of the data and the data storage inside the memory. The project report consists of the assumptions and constraints for the storage manager, the properties and the relations of the storage structures, and the DDL & DML operations that are supported by the storage manager.

2 Assumptions & Constraints

2.1 Assumptions

2.1.1 Base Assumptions

This subsection contains the base assumptions that are already mentioned in the project description.

- Page Size = 1024 Bytes
- File Size = 20488 Bytes
- The content of the File Header:
 - # of Total Pages
 - # of Active Pages
- The content of the Page Header:
 - Boolean: isActive -> States if the page is active/deleted.
 - # of Total Records
 - # of Active Records
- The content of the Record Header:
 - Boolean: isActive -> States if the record is active/deleted.
 - # of Fields
 - Primary Key (Record ID)
- Maximum number of fields a type can have = 10
- Maximum length of a type name = 10 characters
- Maximum length of a field name = 10 characters
- User always enters valid input.
- All fields shall be integers. However, type and field names shall be alphanumeric.
- A disk manager already exists that is able to fetch the necessary pages when addressed.

2.1.2 Extra Assumptions

This subsection contains the additional assumptions that are determined by the assignee of the project.

- The contents of the fields are no larger than 4 Bytes.

2.1.3 Added Assumptions

This subsection contains the assumptions added during the second project.

- The system catalogue is a single page which has a dynamical size.

2.2 Constraints

2.2.1 Base Constraints

This subsection contains the base constraints that are already mentioned in the project description.

- The data must be organized in the pages and pages must contain records.
- Storing all pages in the same file is not allowed. Plus, a file must contain multiple pages.
- Although a file contains multiple pages, it must read page by page when it is deleted. Loading the whole file to RAM is not allowed.

2.2.2 Extra Constraints

This subsection contains the additional constraints that are determined by the assignee of the project.

- Two records with the same primary key values cannot be created.
- A file contains the records of only one type.
- Every record of a type has a unique primary key (record id).
- Every file has a unique file name.

- If the user creates a new type, the file name will be in the form of "typeName1.txt". In case of the lack of free space in the file holding the type, the new file that will contain new records of the same type will be named in the form of "typeName2.txt" and this procedure will apply for the next file creations of the same type.

2.2.3 Added Constraints

This subsection contains the constraints added during the second project.

- The sizes of the data storage units are fixed (system catalogue excluded).

3 Storage Structures

The system has is composed of two primary storage structures that are the system catalogue which contains the metadata about the actual data, and the data storage structures that contains the actual data.

3.1 System Catalogue

System catalogue is a file that stores metadata, data about the actual data. The abstraction of the database has been carried out by the system catalogue. The user must first access into this file before making an operation in the database. The system catalogue consists of only one page which holds the records consisting of the abstraction of the data within the files.

- System Catalogue Header

– # of Files: 4 Bytes

4 Bytes

of Files

- System Catalogue Content

The system catalogue consists of a single page which holds the records corresponding to the abstraction of each prospected files to be created in case of necessity. The content of each record is as follows:

- Boolean: isActive : 1 Byte
-> States if the file is active/deleted.
- File Name: 16 Bytes
- Type Names: 10 Bytes
- # of Fields of The Records of The Corresponding File: 4 Bytes

31 Bytes Per Record (+4 Bytes due to the file processing device used)

isActive	File Name	Type Name	# of Fields
boolA	FileA	TypeA	#A
boolB	FileB	TypeB	#B
boolC	FileC	TypeC	#C

Table 1: The Structure of The System Catalogue

3.2 Data Storage Units

1. Files: 20488 Bytes

- File Header: 8 Bytes
 - # of Total Pages: 4 Bytes
 - # of Active Pages: 4 Bytes

# of Total Pages	# of Active Pages
------------------	-------------------

- File Content: A file contains maximum 20 pages.

$$20 \times 1024 = 20480 \text{ Bytes per File}$$

Page #1
Page #2
..
Page #20

2. Pages: 1024 Bytes

- Page Header: 9 Bytes
 - Boolean isActive: 1 Byte
 - # of Total Records: 4 Bytes
 - # of Active Records: 4 Bytes

isActive	# of Total Records	# of Active Records
----------	--------------------	---------------------

- Page Content: 1015 Bytes

Record #1
Record #2
..
Record #N

3. Records: $9 + (\# \text{ of Fields (Primary Key Excluded)} \times 4)$ Bytes

- Record Header: 9 Bytes
 - Boolean isActive: 1 Byte

- # of Fields: 4 Bytes
- Primary Key (Record ID): 4 Bytes

isActive	# of Fields	Primary Key
----------	-------------	-------------

- Record Content: A record contains maximum 10 fields. (Primary Key field is considered as the part of the record header)

(# of Fields) x 4 Bytes per Record

Field#1	Field#2	...	Field#10
---------	---------	-----	----------

4 Operations

4.1 DDL Operations

DDL stands for the abbreviation of Data Definition Language which consists of the operations that are used to make creations, deletions, modifications, etc. on the database schema.

4.1.1 Create a Type

Algorithm 1 Create a Type

```
declare newType
//initialize the fields of the newType
newType.typeName ← input
newType.primaryKey ← input
newType.numOfFields ← input
for  $i = 1$  to  $newType.numOfFields$  do
  | newType.field#i ← input
end
systemCatalogue.append(newType)
systemCatalogue.numOfFiles++
```

4.1.2 Delete a Type

Algorithm 2 Delete a Type

```
typeName ← input
listOfIndices ← getList(typeName)
if listOfIndices.size() == 0 then
    System.out.println("Type Not Found!")
    break
end
else
    for i = listOfIndices.size() to 1 do
        index ← listOfIndices[i-1]
        systemCatalogue.types.remove(index)
        systemCatalogue.header.numOfFiles--
        systemCatalogue.files.delete(typeName#i)
    end
end
```

4.1.3 List All Types

Algorithm 3 List All Types

```
if systemCatalogue.header.numOfFiles == 0 then
    System.out.println("No Types Exist!")
    break
end
else
    numOfFiles ← systemCatalogue.header.numOfFiles
    for i = 1 to numOfFiles do
        curLine ← "systemCatalogue.txt".nextLine
        print curLine[1]
    end
end
```

4.2 DML Operations

DML stands for the abbreviation of Data Manipulation Language which consists of the operations that are used to make creations, deletions, modifications, etc. on the data itself.

4.2.1 Create a Record

Algorithm 4 Create a Record

```
declare newRecord
newRecord.content.typeName ← input
newRecord.content.numOfFields ← systemCatalogue.getNumOfFields(typeName)
for  $i = 1$  to  $newRecord.numOfFields$  do
    | newRecord.field#i ← input
end
if  $systemCatalogue.header.numOfFiles == 0$  then
    | newFile ← new File("newRecord.content.typeName#i.txt")
    | newPage ← new Page(), newPage.records.add(newRecord)
    | newPage.numOfRecords++
    | newFile.pages.add(newPage), newFile.numOfPages++
    | systemCatalogue.files.add(newFile), systemCatalogue.numOfFiles++
end
else
    | curFile ← systemCatalogue.header.firstFile
    | while  $curFile.nextFile \neq NULL$  do
        | if  $curFile.getType().equals(newRecord.typeName) \&\& curFile.numOfFreeBytes$ 
        |  $\neq 0$  then
            | foreach  $curPage$  in  $curFile$  do
                | foreach  $curRecord$  in  $curPage$  do
                    | if  $curRecord == NULL$  then
                        | | curRecord ← newRecord, curPage.numOfRecords++
                        | | return
                    | end
                | end
            | end
        | end
    | end
    | if  $systemCatalogue.numOfFiles < 20$  then
        | newFile ← new File("newRecord.content.typeName#i.txt")
        | newPage ← new Page(), newPage.records.add(newRecord)
        | newPage.numOfRecords++
        | newFile.pages.add(newPage), newFile.numOfPages++
        | systemCatalogue.files.add(newFile), systemCatalogue.numOfFiles++
    | end
    | else
        | System.out.println("Not Enough Space!")
    | end
end
end
```

4.2.2 Delete a Record

Algorithm 5 Delete a Record

```
primaryKey ← input
typeName ← input
if systemCatalogue.header.numOfFiles == 0 then
  | System.out.println("No Records Exist!"), break
end
else
  curFile ← systemCatalogue.header.firstFile
  while true do
    while !curFile.typeName.equals(typeName) do
      if curFile.nextFile != NULL then
        | curFile ← curFile.nextFile
      end
      else
        | return
      end
    end
    foreach curPage in curFile do
      foreach curRecord in curPage do
        if curRecord.primaryKey == primaryKey then
          curPage.numOfRecords--
          curRecord = NULL
          if curPage.numOfRecords == 0 then
            curFile.pages.remove(curPage)
            curFile.numOfPages--
            if curFile.numOfPages == 0 then
              systemCatalogue.files.remove(curFile)
              systemCatalogue.header.numOfFiles--
            end
          end
          return
        end
      end
    end
    curFile ← curFile.nextFile
  end
end
```

4.2.3 Search for a Record

Algorithm 6 Search a Record (by Primary Key)

```
primaryKey ← input
typeName ← systemCatalogue.getTypeName(primaryKey)
curFile ← systemCatalogue.header.firstFile
while true do
    while !curFile.typeName.equals(typeName) do
        if curFile.nextFile != NULL then
            curFile ← curFile.nextFile
        end
        else
            | return
        end
    end
    foreach curPage in curFile do
        foreach curRecord in curPage do
            if curRecord.primaryKey == primaryKey then
                | return curRecord
            end
        end
    end
    curFile ← curFile.nextFile
end
```

4.2.4 Update a Record

Algorithm 7 Update a Record (by Primary Key)

```
primaryKey ← input
isFoun ← false if systemCatalogue.header.numOfFiles == 0 then
|   System.out.println("No records exist!") return
end
else
|   while isFound==false do
|       while !curFile.typeName.equals(typeName) do
|           if curFile.nextFile != NULL then
|               |   curFile ← curFile.nextFile
|           end
|           else
|               |   return
|           end
|       end
|       foreach curPage in curFile do
|           foreach curRecord in curPage do
|               if curRecord.primaryKey == primaryKey then
|                   |   updatedRecord ← curRecord
|                   |   isFound ← true
|               end
|           end
|       end
|       if isFound == false then
|           |   curFile ← curFile.nextFile continue
|       end
|   end
|   numOfFields ← updatedRecord.numOfFields
|   for i=1 to numOfFields do
|       |   updatedRecord.field#i ← input
|   end
end
```

4.2.5 List All Records of A Type

Algorithm 8 List All Records of A Type

```
typeName ← input
if systemCatalogue.header.numOfFiles == 0 then
    System.out.println("No Records Exist!")
    return
end
else
    curFile ← systemCatalogue.header.firstFile
    while curFile.nextFile != NULL do
        if curFile.typeName.equals(typeName) then
            foreach curPage in curFile do
                foreach curRecord in curPage do
                    printRecord(curRecord)
                end
            end
        end
    end
end
end
```

5 Conclusion & Assessment

- A well designed storage manager allows a reliable database storage, efficient operations, and secure access into the database. In this project, the design of the storage manager has been converted into the implementation.
- The content of the storage structures had been designed in order to enable a better view before implementation. By the time the implementation has been proceeded, some unnecessary fields of the headers or uses have been detected and removed out of the project.
- For further information about the headers, some fields might have been added such as remaining bytes, timestamp fields, etc. ,but in order to provide a less complex implementation the fields inside the headers have been simplified as much as possible.
- The files containing a single type of record allow the administrators to easily handle the storage of the data by providing a simple and efficient abstraction of data.

PS: Due to the restrictions caused by the use of algorithm block in LaTeX, a blank page has been automatically generated when the page size has not been enough for the algorithm block.

5.1 Changes

1. File header has been updated.
2. File size has been updated.
3. Page header has been updated.
4. Page size has been updated into a fixed size of 1024 Bytes.
5. Record header has been updated.
6. Added assumptions constraints have been denoted in the corresponding sections
7. System catalogue header has been updated. System catalogue has been designed to be a single page of dynamical size.

8. Create type operation pseudo-code has been updated.