BOĞAZİÇİ UNIVERSITY

March 10, 2019

CMPE 321

-Project 1-

Database Storage Manager Design

Fahri Can Şanlı

2015400096

Spring, 2019

Contents

1 Introduction

This project is the first project of CmpE 321 course for Spring 2019.

It is about to design a database storage manager which offers some **Data Definition Language** and **Data Manipulation Language** operations. The design will have a System Catalogue which has pages and records and Data files in which there are pages with records and their related field values.

- DDL operations are:

- Create a record type
- Delete a record type
- List all the existing types

- DML operations are:

- Create a record
- Delete a record
- Search for a record by primary key
- List all the existing records of a type

In the next sections, all necessary explanations will be given for each operation and pseudo-code will be provided for each operation.

2 Assumptions and Constraints

This section is about giving necessary assumptions and constraints that are made for DBMS and they are as following

2.1 System Catalogue

- System will have a System Catalogue named "systemCat.txt".
- System shall not allow to create another system catalogue or delete the existing one.
- System catalogue shall composed of pages.
- Each system catalogue page should contain a page header and records with their field names.
- Each system catalogue page should contain at most 12 records.

2.2 Data Files

- Data files should composed of pages.
- Data files should contain at most 10 pages.
- Data files should have a name "datafile_type_id.txt".
- When creating a new file for a type a unique id shall be assigned by function.

2.3 Data File Pages

- Data file pages should contain at most 30 records with their field values.
- Data file pages should contain only one type of record. It should not contain mixture of record types.
- Data file page will have a name ”page_type_id.txt”.
- Data file pages should have size of 1400 bytes.
- When creating a new page for a type a unique id shall be assigned by function.

2.4 Records

- Field names shall be restricted to at most 10 bytes long.
- Field names shall be composed of alphanumeric and case-sensitive characters.
- Field values shall be integer.
- Each record should have 10 fields and a header. More fields shall not be allowed to create.
- Each record header shall have an id, a boolean value and an integer.
- Duplicate field name shall not be allowed.
- Duplicate record type shall not be allowed.
- Unassigned field names shall be null and unassigned field values shall be 0.

2.5 System

- System should have UTF-8 format.
- Each character is 1 byte.
- Each integer is 4 bytes.

3 Storage Structures

This design will have two important components which are **System Catalogue** and **Data Files**.

3.1 System Catalogue

The System Catalogue is the most important part of this design. It is used to store all the metadata. It contains multiple pages with records with fixed size

Its design is as follows :

3.1.1 Page Header

- Page ID - 4 bytes
- Pointer to the next page - 4 bytes
- # of types that are included in the page - 1 byte

3.1.2 Records

- Record Header
  - Type Name - 10 bytes
  - # of fields that are used - 1 bytes
  - isTypeDeleted - 1 byte
- Fields
  - Field Name 1 - 10 bytes
  - Field Name 2 - 10 bytes
  - ....
  - ....
  - Field Name 10 - 10 bytes

Page ID			Pointer to the next page		# of types registered	
Type Name #1	# of fields used	isTypeDeleted	Field Name 1	Field Name 2	....	Field Name 10
Type Name #2	# of fields used	isTypeDeleted	Field Name 1	Field Name 2	....	Field Name 10
....	....	....	....	....	....	....
....	....	....	....	....	....	....
Type Name #12	# of fields used	isTypeDeleted	Field Name 1	Field Name 2	....	Field Name 10

Table 1: Page demonstration for a page of System Catalogue

3.2 Data Files

In this design of DBSM, data files are the second required part. They are actual data of the database. Each data file can store at most 10 pages and those pages are restricted to be consist of records of one type. Records are instances of a type in the system catalogue. Therefore, fields are filled with actual values and these values must be integer.

3.2.1 File Header

- File ID - 4 bytes
- # of pages that are registered - 1 bytes
- File type - 10 bytes
- A pointer to the next file - 4 bytes

3.2.2 Page Header

- Page ID - 4 bytes
- # of records that are registered - 1 bytes
- Page type - 10 bytes
- A pointer to the next page - 4 bytes

3.2.3 Records

- Record Header
  - Record ID - 4 bytes
  - isEmpty - 1 byte
- Fields
  - Field Value 1 - 4 bytes
  - Field Value 2 - 4 bytes
  - ....
  - ....
  - Field Name 10 - 4 bytes

File ID	# of pages registered	File Type	Pointer to the next file
Page 1			
Page 2			
....			
....			
Page 10			

Table 2: Data File Illustration

Page ID		# of records registered	Page Type		Pointer to the next file
Record ID 1	isEmpty	Field Value 1	Field Value 2	....	Field Value 10
Record ID 2	isEmpty	Field Value 1	Field Value 2	....	Field Value 10
....	....	....	....	....	....
....	....	....	....	....	....
Record ID 30	isEmpty	Field Value 1	Field Value 2	....	Field Value 10

Table 3: Data File Page Illustration

4 Algorithms

Note: In the following algorithms, it is assumed that getFirstPageHeader function returns the first page header of system catalogue and after ”**goto pageHeader.nextPage**” step, we are able to reference current page.

4.1 DDL Operations

DDL operations are generally related to System Catalogue as we know.

4.1.1 Create A Type

Algorithm 1: Creating Data Type

```
procedure createType(string nameOfType, int numOfFields)
open("systemCat.txt")
pageHeader ← getFirstPageHeader()
while pageHeader.registeredTypes==12 do
    goto pageHeader.nextPage
end
pageHeader.registeredTypes++
if pageHeader.registeredTypes == 12 then
    newPage ← createNewPage()
    pageHeader.nextPage ← newPage.pageHeader.pageID
end
createNewFile(nameOfType)
newRecord ← createNewRecord(nameOfType,numOfFields)
for 0 to numOfFields do
    fieldname ← scan(input)
    newRecord.fields[i] ← fieldName
end
```

4.1.2 Delete A Type

Algorithm 2: Deleting Data Type

```
procedure deleteType(string nameOfType)
fileToBeDeleted ← findFile(nameOfType)
deleteFile(fileToBeDeleted)
open("systemCat.txt")
foreach page in "systemCat.txt" do
    foreach record in page do
        if record.isTypeDeleted == 0 and record.nameOfType == nameOfType then
            record.isTypeDeleted ← 1
            return
        end
    end
end
```

Note: *findFile finds all data files of this type and deleteFile deletes those files and records of them*

4.1.3 List All Types

Algorithm 3: Listing All Data Types

```
procedure listAllTypes()
open("systemCat.txt")
foreach page in "systemCat.txt" do
    foreach record in page do
        if record.isTypeDeleted != 1 then
            print record.nameOfType
        end
    end
end
```

4.2 DML Operations

DML operations are generally related to data files.

4.2.1 Create A Record

Algorithm 4: Creating A New Record of A Type

```
procedure createRecord(string nameOfType)
numOfFields ← getNumOfFields(nameOfType, "systemCat.txt")
file ← getFile(nameOfType)
open(file)
pageHeader ← getFirstPageHeader()
while pageHeader.registeredRecords == 30 do
    goto pageHeader.nextPage
end
pageHeader.registeredRecords++
if file.registeredPages != 10 and pageHeader.registeredRecords == 30 then
    newPage ← createNewPage(nameOfType)
    pageHeader.nextPage ← newPage.pageHeader.pageID
    file.registeredPages++
end
Else if file.registeredPages == 10 and pageHeader.registeredRecords == 30 then
    newFile ← createNewFile(nameOfType)
    file.nextFile ← newFile.fileHeader.fileID
    newPage ← createNewPage(nameOfType)
    newfile.registeredPages++
end
```

```
foreach record in page do
    if record.isEmpty == 1 then
        record.isEmpty ← 0
        for 0 to numOfFields do
            record.fieldValue[i] ← scan(input)
        end
        return
    end
end
recordNewId ← getNewId()
recordNew ← createNewRecord(nameOfType, recordNewId)
for 0 to numOfFields do
    recordNew.fieldValue[i] ← scan(input)
end
```

4.2.2 Delete A Record

Algorithm 5: Deleting An Existing Record

```
procedure deleteRecord(string nameOfType, int recordID)
file ← findFile(nameOfType)
page ← findPage(file, recordID)
recordToBeDeleted ← findRecord(page, recordID)
numOfFields ← getNumOfFields(nameOfType, ”systemCat.txt”)
recordToBeDeleted.isEmpty ← 1
for 0 to numOfFields do
    recordToBeDeleted.fieldValues[i] ← 0
end
page.registeredRecords–
if page.registeredRecords == 0 then
    deletePage(page)
    file.registeredPages–
    if file.registeredPages == 0 then
        deleteFile(file)
    end
end
```

Note: *deleteFile and deletePage functions handle the pointer arrangement after delete operation*

4.2.3 Search A Record

Algorithm 6: Searching For A Record By Its Primary Key

```
procedure searchRecord(string nameOfType, int recordID)
file ← findFile(nameOfType)
open(file)
foreach page in file do
    foreach record in page do
        if record.isEmpty == 0 and record.recordID == recordID then
            return record
        end
    end
end
```

4.2.4 List All Records of A Type

Algorithm 7: Listing All Existing Records of A Specific Type

```
procedure listRecords(string nameOfType)
file ← findFile(nameOfType)
open(file)
foreach page in file do
    foreach record in page do
        if record.isEmpty == 0 then
            print record
        end
    end
end
```

5 Conclusion & Assessment

In this project, a database storage manager is designed and documented. In this design, each structure's size is fixed. For instance, each page has size of 1400 bytes. However, the system does not use all of the allocated space. Data file page uses 1369 bytes and system catalogue page uses 1353 bytes. The logic behind this is to make system more reliable. Yet, this logic causes the system waste unnecessary memory. Insertion of new records and pages are done without any specific order. It is good for insertion operation, but it makes search operation slower and harder. Therefore, it could be added to this design. Also note that, in this design isEmpty variable is used for deleting a record. When a record is deleted, isEmpty variable of that record becomes True and a page is not deleted completely until all records of that page have True value for isEmpty variable. Besides, in my opinion deleting records from System catalogue during a database update or backup is better.

To put in a nutshell, this design has some cons and pros. Since, it is the first part and a design part, some mistakes might be made. In the implementation part, everything will be clearer and those mistakes are going to be fixed.