BOĞAZİÇİ UNIVERSITY

CMPE 321

PROJECT 1

Summer 2017

Storage Manager Design

Mustafa Enes ÇAKIR

July 10, 2017

Contents

1 Introduction

A storage manager is a program that controls how the memory will be used to save data to increase the efficiency of a system. In this project, I have designed a Storage Manager System without error checking. This document consists of the design details for a simple database management system, including data structures, explanations and implementations (in pseudo-code) of the available database operations. According to my design, there is a system catalog which stores metadata, and multiple data files that store actual data. I have constructed the system in a way that to allocate a different page for each type. The storage manager will keep the datas in typeName.txt and SysCat.cat will be the guide for the system catalogue. The records in a file are divided into unit collections (pages) in abstraction.

2 Assumptions & Constraints

- Page will be 1400 bytes.
- Every character is 1 byte.
- All of the field values are integer.
- A data type can contain 10 fields provided by user exactly. More fields is not allowed, yet if it contains less field, the remaining fields are considered as null.
- A page will contain at most one type of record.
- Field names are at most 12 characters long.
- No two fields of a data type have the same name.
- Data files have the format ¡type-name¿.txt.
- A data file can contain multiple pages.
- System catalog file has the name SysCat.txt.

3 Data Structures

This design contains two main components which are System Catalogue and Data Files.

3.1 System Catalogue

System catalogue is responsible for storing the metadata. It's a blueprint for datas. It has multiple pages. Each record has a fixed size. Each type has 10 fields. Therefore, the space of each record in the system catalog is 133 bytes. So 10 data type records can be stored in a page.

- Page Header (8 bytes)
  - Page ID (4 bytes)
  - # of Records (4 bytes)
- Record (133 bytes)
  - Record Header (13 bytes)
    - Type Name (12 bytes)
    - # of Fields (1 byte)
  - Field Names (10 x 12 = 120 bytes)

Page ID			# of Records		
Type Name 1	# of Fields	Field Name 1	Field Name 2	...	Field Name 10
Type Name 2	# of Fields	Field Name 1	Field Name 2	...	Field Name 10
...	...	...	...	...	...
Type Name 10	# of Fields	Field Name 1	Field Name 2	...	Field Name 10

Table 1: a Page of a System Catalogue *(Starting with the Page Header)*

3.2 Data Files

Data files store actual datas. In this storage manager system, data files are separated into the number of types. Each data file can store one type of record and is named with that type. *"hamster.txt"* can store only the records of type *hamster* for example. These records can be thought as instances of a type in the system catalog. Therefore the fields of the records are actual values. Here, all the values are integer. A record in a data file has a space of 45 bytes. Each page in a data file can store 30 records at most.

3.2.1 Pages

Page headers store information about the specific page it belongs to.

- Page Header (13 bytes)
  - Page ID (4 bytes)
  - Pointer to Next Page (4 bytes)
  - # of Records (4 bytes)
  - isEmpty (1 byte)
- Records (a Record = 45 bytes)

3.2.2 Records

- Record Header (5 bytes)
  - Record ID (4 bytes)
  - isEmpty (1 bytes)
- Fields (10 x 4 = 40 bytes)

Page ID	Pointer to Next Page		# of Records		isEmpty
Record ID 1	isEmpty	Field 1	Field 2	...	Field 10
Record ID 2	isEmpty	Field 1	Field 2	...	Field 10
...	...	...	...	...	...
Record ID 30	isEmpty	Field 1	Field 2	...	Field 10

Table 2: Page of a Data File *(Starting with the Page Header)*

4 Algorithms

4.1 DDL Operations

Database Design Language *(DDL)* operations are generally is related to system catalogue. As we have declared.

4.1.1 Create a type

Algorithm 1: Creating Data Type

```
declare dataType
nameOfType ← User Input
numberOfFields ← User Input
dataType.push(nameOfType, numberOfFields)
for 0 to numberOfFields do
    nameOfField ← User Input
    dataType.push(nameOfField)
end
for numberOfFields to 10 do
    dataType.push(NULL)
end
file ← open("SysCat.txt")
file.push(dataType)
createFile(nameOfType.txt)
```

4.1.2 Delete a type

Algorithm 2: Deleting Data Type

```
nameOfType ← User Input
deleteFile(nameOfType.txt)
file ← open("SysCat.txt")
foreach page in file do
    foreach record in page do
        if record.isEmpty = 0 and record.typeName = nameOfType
          then
            record.isEmpty ← 1
            break
        end
    end
end
```

4.1.3 List all types

Algorithm 3: Listing Data Types

```
file ← open("SysCat.txt")
foreach page in file do
    foreach record in page do
        if record.isEmpty = 0 then
            print(record.typeName)
        end
    end
end
```

4.2 DML Operations

Database Manipulation Language *(DML)* are generally is related to data files.

4.2.1 Create a record

Algorithm 4: Creating Record

```
recordType ← User Input
fieldNum ← getFieldNum(recordType, "SysCat.txt")
file ← open(recordType.txt)
foreach currentPage in file do
    if currentPage.numOfRecords != 30 then
        page ← currentPage
        break
    end
end
foreach record in page do
    if record.isEmpty = 1 then
        record.isEmpty ← 0
        for i ← 0 to fieldNum do
            record[i + 2] ← User Input
        end
        page.numOfRecords++
        break
    end
end
```

4.2.2 Delete a record

Algorithm 5: Deleting Record

```
recordType ← User Input
recordID ← User Input
file ← open(recordType.txt)
foreach page in file do
    foreach record in page do
        if record.isEmpty = 0 and record.id = recordID then
            record.isEmpty ← 1
            page.numOfRecords–
            break
        end
    end
end
```

4.2.3 Search for a record

Algorithm 6: Deleting Record

```
recordType ← User Input
recordID ← User Input
file ← open(recordType.txt)
foreach page in file do
    foreach record in page do
        if record.isEmpty = 0 and record.id = recordID then
            return record
        end
    end
end
```

4.2.4 List all records of a type

Algorithm 7: Listing All Records

```
declare records
recordType ← User Input
file ← open(recordType.txt)
foreach page in file do
    foreach record in page do
        if record.isEmpty = 0 then
            records.push(record)
        end
    end
end
return records
```

5 Conclusions & Assessment

In this documentation a storage manager design is proposed where size of each structure is fixed. This creates an inefficiency in terms of memory usage while it makes the storage manager easier to implement. I allocated 1400 Bytes space for pages however the system only uses 1363 Bytes, this is done for Reliability. Also note that, the pages and records are inserted to storage manager linearly without any specific order. This makes searching slower whereas it makes insertion faster. It's faster when listing records.
This restricts the user to some extent, however it makes the storage manager faster since the length controls are unnecessary, no checking for identical key is needed.

To sum up, this design has its own ups and downs just like every design. Since it is kept as a simple one, it is easy to modify and improve. Hence, implementing it would also be easier with necessary modifications that can be realized on the run.