BOĞAZİÇİ UNIVERSITY

CMPE 321

PROJECT 1

Storage Management System Design

Spring 2020

Yunus Kardaş

March 27, 2020

Contents

1 Introduction

In this project, I am expected to design a storage manager system that supports DDL operations and DML. There should be a system catalogue which stores metadata and multiple data files that store the actual data. This document explains my design by showing my assumptions, constraints, data structures and explaining the algorithms behind the DDL and DML operations in pseudocode.

2 Assumptions & Constraints

- System catalog file has the name SysCat.txt.
- The system shall not allow to create more than one system catalog file or delete an existing one.
- Every character is 1 byte and every integer is 4 bytes.
- A page will be 1400 bytes.
- 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 cannot hold records of 2 different types, it has to hold at most one type.
- Two fields of a record cannot have the same name.
- Field names are at most 10 characters long.
- No two fields of a data type have the same name.
- Data files have the format type-name.txt.
- A record type can contain at most 10 fields provided by the user. If it contains less, remaining fields considered as null.
- A file can contain multiple pages.

3 Storage Structures

This storage manager contains two components which are System Catalogue and Data Files.

3.1 System Catalogue

System catalogue is the main file of the store manager. It is responsible for storing the metadata. Any change that can be done in the system via this file. It has the name ‘SysCat.txt’. It has multiple pages.

- Page Header (8 bytes)
  - Page ID (4 bytes)
  - # of Records (4 bytes)
- Record (115 bytes)
  - Record Header (15 bytes)
    - Type Name (10 bytes)
    - # of Fields (4 bytes)
    - Deletion Status (isDeleted)(1 byte)
  - Field Names (10 x 10 = 100 bytes)

Page ID	# of Records
Record Header	Field Names

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

3.2 Data Files

Data files store actual datas. Each data file can store at most one type of record. Data files have the name type-name.txt. Each page in a data file can store at most 32 records.

3.2.1 Pages

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

- Page Header (13 bytes)
  - Page ID (4 bytes)
  - # of Records (4 bytes)
  - isEmpty (1 byte)
  - Pointer to Next Page (4 bytes)
- Records (a Record = 46 bytes)

3.2.2 Records

- Record Header (6 bytes)
  - Record ID (4 bytes)
  - isEmpty (1 bytes)
- Fields (10 x 4 = 40 bytes)

<table>
<tr><td>Page ID</td><td colspan="2">Pointer to Next Page</td><td colspan="2"># of Records</td><td>isEmpty</td></tr>
<tr><td colspan="3">Record Header</td><td colspan="3">Field Names</td></tr>
<tr><td>Record ID 1</td><td>isEmpty</td><td>Field 1</td><td>Field 2</td><td>...</td><td>Field 10</td></tr>
<tr><td>Record ID 2</td><td>isEmpty</td><td>Field 1</td><td>Field 2</td><td>...</td><td>Field 10</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>Record ID 30</td><td>isEmpty</td><td>Field 1</td><td>Field 2</td><td>...</td><td>Field 10</td></tr>
</table>

4 Operations

4.1 DDL Operations

4.1.1 Create a type

```
function creatType
declare recordType
recordType ← User Input
recordType.numberOfFields ← User Input
for integer i=0 to recordType.numberOfFields do
    recordType.fields[i].name ← User Input
end
if recordType.numberOfField is smaller than 10 then
    for int i=recordType.numOfFields+1 to 10 do
        recordType.fields[1].name ← (NULL)
    end
end
file ← open(”SysCat.txt”)
write file recordType
file.pageHeader.numberOfRecords++
ccreateFile(‘recordType.name.txt’)
```

4.1.2 Delete a type

```
function deleteType
file ← findFile(recordsTypeName)
delete file
catalogue ← open(‘SysCat.txt’)
deleteFile(nameOfType.txt)
file ← open(”SysCat.txt”)
for each page in catalogue do
    for each record in page do
        if record.typeName = recordTypeName then
            record.isDeleted ← 1
        end
    end
end
```

4.1.3 List all types

```
function listAllTypes
declare types
file ← open(”SysCat.txt”)
for each page in file do
    for each record in page do
        if record.isDeleted=0 then
            types.push(record.typeName)
        end
    end
end
```

4.2 DML Operations

4.2.1 Create a record

```
function createRecord
recordType ← User Input
file ← open(‘SysCat.txt’)
numOfFields ← file.recordType.numberOfFields
recordFile ← open(‘recordType.txt’)
for each currentPage in recordFile do
    if page.pageHeader.numberOfRecords ≤ 31 then
        lastPage ← page
    end
end
lastPage.pageHeader.numberOfRecords++
for  each record in lastPage do
    if record.isEmpty = 1 then
        record.isEmpty ← 0
        for i = 0 to numOfFields do
            record[i] ← User Input
        end
        record.isEmpty ← 0
    end
end
```

4.2.2 Delete a record

```
function deleteRecord
recordType ← User Input
primaryKey ← User Input
file ← open(recordType.txt)
for  each page in file do
    for each record in page do
        if record.isDeleted = 0 and record.id = primaryKey then
            page.pageHeader.numberOfRecords - -
            record.isDeleted ← 1
            record.isEmpty ← 1
        end
    end
end
```

4.2.3 Search for a record

```
function searchRecord
declare searchedRecord
recordType ← User Input
primaryKey ← User Input
file ← open(recordType.txt)
for each page in a file do
    for each record in a page do
        if record.id = primaryKey and record.isDeleted = 0 then
            searchedRecord ← record
        end
    end
end
return searchedRecord
```

4.2.4 Update a record

```
function updateRecord
declare updatedRecord
recordType ← User Input
primaryKey ← User Input
updatedRecord ← User Input
file ← open(recordType.txt)
for each page in a file do
    for each record in a record do
        if record.id = primaryKey and record.isDeleted = 0 then
            record ← updatedRecord
        end
    end
end
```

4.2.5 List all records of a type

```
function listRecords
declare allRecords
recordType ← User Input
file ← open(recordType.txt)
for each page in file do
    for each record in page do
        if record.isDeleted textbfand record.isEmpty = 0 then
            allRecords.push(record)
        end
    end
end
return allRecords
```

.

5 Conclusions & Assessment

In this project, I have designed a simple storage manager which has a system catalogue file and data files. In my design each file can hold at most one record type and can have at most 100 pages. This makes accessing a record with its primary key faster but insertion is slower since we have to access a specific page to insert a record. Since we didn't do any error checking, if a user enters a wrong input, this storage manager cannot handle it. Also because of fixed page structure we lose some memory that we might be able to use in storing more data. To sum up, this is a really simple storage manager design and it has its own pros and cons. But mostly, it is very efficient while accessing a record but not so much while insertion. But we can modify this design and improve it.