

BOGAZICI UNIVERSITY, ISTANBUL, TURKEY  
CMPE321, SPRING 2020

## PROJECT 1

---

# Designing Storage Manager System

---

Bekir Yıldırım - 2014400054

31 March 2020

Date Performed ..... 31 March, 2020  
Instructor ..... Tunga Gungor

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Assumptions &amp; Constraints</b>	<b>3</b>
2.1	Assumptions . . . . .	3
2.1.1	Type . . . . .	3
2.1.2	System Catalogue . . . . .	4
2.1.3	Page . . . . .	4
2.1.4	File . . . . .	4
2.1.5	Record . . . . .	4
2.1.6	Field . . . . .	4
2.1.7	Users . . . . .	4
2.1.8	Disk Manager . . . . .	4
2.2	Constraints . . . . .	5
2.2.1	Type . . . . .	5
2.2.2	System Catalogue . . . . .	5
2.2.3	Page . . . . .	5
2.2.4	File . . . . .	5
2.2.5	Record . . . . .	5
2.2.6	Field . . . . .	5
<b>3</b>	<b>Storage Structures</b>	<b>6</b>
3.1	System Catalogue . . . . .	6
3.1.1	# of Pages(4 bytes) . . . . .	6
3.1.2	Page Header(8 bytes) . . . . .	6
3.1.3	Record(115 bytes) . . . . .	6
3.2	Data Files . . . . .	6
3.2.1	Pages(2048 bytes) . . . . .	6
3.2.2	Records(46 bytes) . . . . .	7
<b>4</b>	<b>System Design</b>	<b>8</b>
4.1	System Catalogue . . . . .	8
4.2	Page Design & Page Header . . . . .	8
<b>5</b>	<b>Operations</b>	<b>9</b>
5.1	DDL Operations . . . . .	9
5.1.1	Create a type . . . . .	9
5.1.2	Delete a type . . . . .	9
5.1.3	List all types . . . . .	10
5.2	DML Operations . . . . .	11
5.2.1	Create a record . . . . .	11
5.2.2	Delete a record . . . . .	12
5.2.3	Search for a record with Primary Key . . . . .	12
5.2.4	Update for a record with Primary Key . . . . .	13
5.2.5	List all records of a type . . . . .	13
<b>6</b>	<b>Conclusions &amp; Assessment</b>	<b>14</b>

# 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. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database. Can be thought as the interface between the DBMS and all the "physically" at the low levels. Storage manager is responsible for retrieving a record, and the other parts of the DBMS are only concerned with the records, not with files, pages, disk and so on. In this project, I am expected to design a storage manager system that supports DDL and DML operations. There should be a system catalogue which stores metadata and multiple data files that store the actual data.

## 1. DDL Operations

- Create a type
- Delete a type
- List all types

## 2. DML Operations

- Create a record
- Delete a record
- Search for a record by primary key
- Update record by primary key
- List all record for a type

This documents explains my design by showing my assumptions, constraints, storage structures and explaining the algorithms behind the DDL and DML operations in pseudo code. In according to assumptions and constraints, our storage structures and algorithm are arranged. The storage manager will keep according datas in "*typeName.txt*" and "*System-Catalogue.txt*" will be the system catalogue. The records in a file are divided into unit collections(pages) in abstraction.

# 2 Assumptions & Constraints

Before the design of a storage manager system, one needs to determine the constraints on which the design decisions will be made, since it is not possible make a design without any limits/constraints. Also, we should make some assumptions to construct useful storage structures and algorithm.

## 2.1 Assumptions

### 2.1.1 Type

- UTF-8 standard is used in the system, it means a char equal 1 byte.
- A data type can contain 10 fields provided by user exactly. Field values can only be integer as stated in the description.

- More fields are not allowed, yet if it contains less field, the remaining fields will be null.
- I choose the separate file style. Which means there will be a file for each entity type.
- Duplicate names of data types are not allowed.
- Type names shall be alphanumeric.

### **2.1.2 System Catalogue**

- File name of the System Catalogue is "*SystemCatalogue.txt*".
- System Catalogue file cannot be deleted by any user.
- The system shall not allow to create more than one system catalogue file or delete an existing one.

### **2.1.3 Page**

- Page will be 2048 bytes.
- Page header stores; id of the page, type of records in that page, pointer address to next page in the file, number of record in that page, if that page is empty or not and maximum size of the page(which is 2048 bytes).

### **2.1.4 File**

- Data files have the format "*typeName.txt*".
- A data file can contain multiple pages.

### **2.1.5 Record**

- Record consist of 10 fields. If it contains less, remaining fields considered as null.
- Record header stores; state of the record(full,empty or deleted) as also integer.

### **2.1.6 Field**

- All of the field values are integers.
- Field names shall be alphanumeric.

### **2.1.7 Users**

- User always enters valid input.

### **2.1.8 Disk Manager**

- A disk manager already exists that is able to fetch the necessary pages when addressed.

## **2.2 Constraints**

### **2.2.1 Type**

- Type names are at most 10 characters long.
- Type names shall be alphanumeric.
- Max length of a type name ( $\geq 8$ )

### **2.2.2 System Catalogue**

- The system shall not allow to create more than one system catalogue file or delete an existing one.

### **2.2.3 Page**

- A page can only contain one type of record.
- Pages must contain records.

### **2.2.4 File**

- Max file size can be 65Kb.
- A file has at most 32 pages.
- Should contain reasonable amount of pages
- Not allowed to store all pages in the same file and a file must contain multiple pages.
- Although a file contains multiple pages, it must read page by page when it is needed. Loading the whole file to RAM is not allowed.
- When a file becomes free due to deletions, that file must be deleted.

### **2.2.5 Record**

- Record consist of 10 fields. If it contains less, remaining fields considered as null.
- The primary key of a record should be assumed to be the value of the first field of that record.
- Records in the files should be stored in ascending order according to their primary keys.

### **2.2.6 Field**

- Field names are at most 10 characters long.
- Field names shall be alphanumeric.
- Max number of fields a type can have ( $\geq 3$ )
- Max length of a type name ( $\geq 8$ )

## 3 Storage Structures

This design contains two main components which are System Catalogue and Data Files. It contains information about how many bytes are made up and how structure they have.

### 3.1 System Catalogue

System catalogue is responsible for storing the metadata. It's a blueprint for data types. Any change that can be done in the system via this file. It has multiple pages. Page header(8 bytes) of system catalogue has information about page id and number of records. number of pages has information about how many pages is included in system catalogue and also has information about record with header and their field names.

#### 3.1.1 # of Pages(4 bytes)

#### 3.1.2 Page Header(8 bytes)

- Page ID (4 bytes)
- # of Records (4 bytes)

#### 3.1.3 Record(115 bytes)

- Record Header(15 bytes)
  - Type Name(10 bytes)
  - # of Fields(not null)(4 bytes)
  - Deletion Status (isDeleted) (1 byte)
- Field Names(10\*10=100 bytes)

### 3.2 Data Files

Data files store current datas. In our designed storage manager system, data files are separated into the number of types. Each data file can store one type of record. Data files have the name "*typeName.txt*". Each page in data file an store at most 43 records.

#### 3.2.1 Pages(2048 bytes)

Page headers store information about the specific page it belongs to and points to next and previous page.

- Page Header (26 bytes)
  - Page ID (4 bytes)
  - Pointer to Next Page (8 bytes)
  - Pointer to Previous Page (8 bytes)
  - # of Records (4 bytes)
  - isEmpty(1 byte)
  - isFull(1 byte)
- Records

### 3.2.2 Records(46 bytes)

- Record Header(6 bytes)
  - Record ID (4 bytes)
  - isEmpty (1 byte)
  - isDeleted (1 byte)
- Record Fields ( $10 \times 4 = 40$  bytes)

## 4 System Design

### 4.1 System Catalogue

# of Pages						
Page Header						
Page ID			# of Records			
Record Header			Field Names			
Type Name 1	# of Fields 1	isDeleted 1	Field Name 1	Field Name 2	...	Field Name 10
Type Name 2	# of Fields 2	isDeleted 2	Field Name 1	Field Name 2	...	Field Name 10
...	...	...	...	...	...	...
Type Name	# of Fields	isDeleted	Field Name 1	Field Name 2	...	Field Name 10

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

### 4.2 Page Design & Page Header

Page Header					
Page ID	Pointer to Previous Page	Pointer to Next Page	# of Records	isEmpty	isFull
Record Header			Fields		
Record Id 1	isEmpty 1	isDeleted 1	Field 1	...	Field 10
Record Id 2	isEmpty 2	isDeleted 2	Field 1	...	Field 10
...	...	...	...	...	...
Record Id	isEmpty	isDeleted	Field 1	...	Field 10

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



## 5 Operations

### 5.1 DDL Operations

Database Design Language (*DDL*) operations are related to System Catalogue most of the time.

#### 5.1.1 Create a type

---

**Algorithm 1:** Creating Data Type

---

```
1 function createType
2 declare recordType
3 catalogue ← open("SystemCatalogue.txt")
4 recordType.name ← User Input
5 recordType.numberOfFields ← User Input
  /* Now this is a for loop to fill field of record */
6 for int i=0 to recordType.numberOfField do
7   recordType.fields[i].name ← User Input
  /* Now this is an if...else conditional loop to fill empty fields of record */
8 if recordType.numberOfField < 10 then
9   for int i=recordType.numberOfFields+1 to 10 do
10    recordType.fields[i].name ← null // Empty fields of record will be
      filled by null.
  /* Now push created recordType to "SystemCatalogue.txt" */
11 catalogue.push(recordType)
12 catalogue.pageHeader.numberOfRecord++
13 createFile("recordTypeName.txt")
14 endFunction
```

---

#### 5.1.2 Delete a type

---

**Algorithm 2:** Deleting Data Type

---

```
1 function deleteType
2 recordType.name ← User Input // Take record will be deleted by user
3 file ← findFile(recordType.name) // Find record file will be deleted by user
4 delete file
5 catalogue ← open("SystemCatalogue.txt")
  /* Now this is a foreach to change delete status of record */
6 foreach page in catalogue do
7   foreach record in page do
8     if record.typeName == recordType.name then
9       record.isDeleted ← 1
10 endFunction
```

---

### 5.1.3 List all types

---

**Algorithm 3:** List All Types

---

```
1 function listAllTypes
2 declare Types
3 catalogue  $\leftarrow$  open("SystemCatalogue.txt")
  /* Now this is a for loop to push all types */
4 foreach page in catalogue do
5   foreach record in page do
6     if record.isDeleted == 0 then
7        $\lfloor$  types.push(record.typeName)
8 return types // Return all types
9 endFunction
```

---

## 5.2 DML Operations

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

### 5.2.1 Create a record

---

**Algorithm 4:** Creating a Record

---

```
1 function createRecord
2 recordType ← User Input
3 catalogue ← open("SystemCatalogue.txt")
4 numberOfFields ← file.recordType.numberOfFields
5 recordFile ← open("RecordType.txt")
6 boolean lastPage ← 0
7 boolean newPage ← 0
  /* Now this is a foreach to put current page as last page */
8 foreach page in RecordFile do
9   if page.pageHeader.numberOfRecords < 43 or page.isFull == 0 then
10     lastPage ← page
11     lastPage.pageHeader.numberOfRecords++
12     lastPage ← 1
13   else
14     create newPage
15     catalogue.numberOfPages++
16     newPage ← 1
  /* Now this is a foreach to fill records in last page and set empty
  status to 0 */
  // If lastPage is not full
17 if lastPage == 0 then
  /* To find appropriate position for record by using BTree */
18   get BTree(lastPage.records)
19   BTree.insert(record)
20   for i ← 0 to numberOfFields do
21     record.fields[i] ← UserInput // Take input for fill fields of record
22   lastPage.pageHeader.numberOfRecords++
23   record.isEmpty ← 0
  // If newPage is created
24 else if newPage == 0 then
25   foreach record in newPage do
26     if record.isEmpty == 1 then
27       for i ← 0 to numberOfFields do
28         record.fields[i] ← UserInput // Take input for fill fields of
          record
29       record.isEmpty ← 0
30       newPage.pageHeader.numberOfRecords++
31 endFunction
```

---

### 5.2.2 Delete a record

---

**Algorithm 5:** Deleting a Record

---

```
1 function deleteRecord
2 recordType  $\leftarrow$  UserInput
3 primaryKey  $\leftarrow$  UserInput // Take input to primaryKey to delete record
4 boolean control  $\leftarrow$  1
5 i  $\leftarrow$  0
6 while control do
7   file  $\leftarrow$  open("RecordType" + i + ".txt")
8   /* Now this is a foreach to delete record with given primary key */
9   foreach page in file do
10    foreach record in page do
11      if record.isDeleted == 0 & record.id == primaryKey then
12        page.pageHeader.numberOfRecord—
13        record.isDeleted  $\leftarrow$  1 // Set record.isDeleted status 1
14        record.isEmpty  $\leftarrow$  1 // Set record.isEmpty status 1
15        control  $\leftarrow$  0
16        if page.pageHeader.numberOfRecord == 0 then
17          delete page
18    i  $\leftarrow$  i+1
19 endFunction
```

---

### 5.2.3 Search for a record with Primary Key

---

**Algorithm 6:** Searching for a Record

---

```
1 function searchRecord
2 declare searchedRecord
3 recordType  $\leftarrow$  UserInput
4 primaryKey  $\leftarrow$  UserInput // Take input to primaryKey to select record
5 boolean control  $\leftarrow$  1
6 i  $\leftarrow$  0
7 while control do
8   file  $\leftarrow$  open("RecordType" + i + ".txt")
9   /* Now this is a foreach to search record with given primary key */
10  foreach page in file do
11    foreach record in page do
12      if record.isDeleted == 0 & record.id == primaryKey then
13        searchedRecord  $\leftarrow$  record // Record is found with primary key.
14        control  $\leftarrow$  0
15    i  $\leftarrow$  i+1
16 return searchedRecord
17 endFunction
```

---

### 5.2.4 Update for a record with Primary Key

---

**Algorithm 7:** Updating for a Record

---

```
1 function updateRecord
2 declare updatedRecord
3 updatedRecord ← UserInput
4 recordType ← UserInput
5 primaryKey ← UserInput // Take input to primaryKey to update record
6 boolean control ← 1
7 i ← 0
8 while control do
9   file ← open("RecordType" + i + ".txt")
10  /* Now this is a foreach to search record with given primary key */
11  foreach page in file do
12    foreach record in page do
13      if record.isDeleted == 0 & record.id == primaryKey then
14        record ← updatedRecord // Record which is finded with
15        primary key is updated.
16        control ← 0
17    i ← i+1
18 endFunction
```

---

### 5.2.5 List all records of a type

---

**Algorithm 8:** Listing All Records for a Type

---

```
1 function listRecords
2 declare allRecords
3 recordType ← UserInput // Take input to list all certain records of a
4   type
5 boolean control ← 1
6 i ← 0
7 while control do
8   file ← open("RecordType" + i + ".txt")
9   if file not exist then
10    break
11   /* Now this is a foreach to search record which is not empty and not
12     deleted */
13   foreach page in file do
14     foreach record in page do
15       if record.isDeleted == 0 & record.isEmpty == 0 then
16         allRecords.push(record)
17     i ← i+1
18 return allRecords // Return all records
19 endFunction
```

---

## 6 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. In my design each file can hold at most one record type and can have at most 32 pages. This make 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 did not do any error checking, if a user enters a wrong input, this storage manager cannot handle it. I decided to use BTree instead of there is no adequate place for show implementation, in second project will implement this algorithm to increase performance for sorting record by primary key .One down for our design could be the performance if the database is large, program might need to resort so much of opening and closing files and pages, which could be a lot of overhead on CPU.

To sum up, this is really simple storage manager design and it has it owns 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. Hence, implementing it would also be easier with necessary modifications that can be realized on the run.