Bogazici University

cmpe 321

Introduction to Database Systems

Implementation of Storage Manager

Author: Ahmet Ege MAHLEC
2016800045

April 23, 2017

1 Introduction

This report is dedicated to implementation of Database Manager which is composed of Storage Manager, Query Manager and Disk Manager (Page Manager). The program which is written by using C++ allows the Data Definition Language (DDL) and Data Manipulation Language (DML) operations which are given below.

DDL Operations

1. Create a type
2. Delete a type
3. List all types

DML Operations

1. Create a record
2. Delete a record
3. Update a record
4. Search for a record via primary key
5. List all records of a type

The program is built by the aid of CMAKE; therefore, it can be built and executed for any operating system which user wishes.

As it is stated above, it is composed of three main levels, namely, Query Manager, Storage Manager and Disk (Page) Manager. While Query Manager listens the user commands, it passes these commands to Storage Manager. Storage Manager uses Page Manager to access the disk page by page. For every operation, whole page is read or written.

The storage manager creates two types of files, namely, system catalog and data files. When user requests to create a new type, the meta data of this type will be stored in the system catalog. In contrast, when user requests

to delete a type, the meta data of this type will be removed from system catalog; moreover data file of this type will be deleted.

The other file type which was created by storage manager is data files. When user requests to create a new type, data file of this type will be created unless it has been created. When user requests to delete a record, record will be erased from the data file of this type. Also, when user request to add a new record, this new record will be added to data file of this type.

2 Assumptions and Constraints

In this section, the design choices will be explained by stating the assumptions and constraints which are used to implement Storage Manager.

- Characters which is not defined in ASCII table will not be given as input.
- Storage manager does not give any service to interact tables with each other.
- Files shall be divided in the pages.
- In order to read or write data, whole page shall be read or written.
- Page size shall be 1 KB.
- Length of the type name shall be at most 16 bytes long.
- Length of the field name shall be at most 16 bytes long.
- Type and field name shall be alphanumeric.
- Every type has at most 10 fields.
- The value of all fields will be type signed integer.
- The value of all fields will be at most 4-byte long.
- User is not allowed to update a type once it has been created. It should be deleted and created again if a record would like to be updated. In this case, all the records will be deleted.

- All the data will be written in little endian format.
- First field of the record will be selected as primary key.
- User will not enter any record whose primary key is same with the record which has already exist in the database.
- User will not enter a type whose name is same with type which has already exist in the database.
- User will not try to delete any record or type which doesn't exist in the database.
- User will not call the query in wrong format.

3 Data Structures

3.1 System Catalog File Format

System catalog is composed of the pages. Every page has header part and data part.

Table 1: System Catalog File Format

Page Header	T#1 Header	T#1 F#1	T#1 F#2	...	T#1 F#10
T#2 Header	T#2 F#1	T#2 F#2	T#2 F#3	...	T#2 F#10
T#5 Header	T#5 F#1	T#5 F#2	T#5 F#3	...	T#5 F#10

Table 2: Page Header Format

PageID (4 byte)	TotalSpace(2 byte)	FreeSpace(2 byte)	Reserved(116 byte)

Table 3: Type Header Format

TypeID (1 byte)	isUsed(1 byte)	NumberOfField(1 byte)	TypeName(16bytes)	Reserved(1 byte)

As it is shown in Table 1, System Catalog File starts with a Page Header and follows by a type header and field name of the types. Since every type has at most 10 fields; if a type is created, 10 x 16 + 20= 180 bytes allocated for this type even if it has less fields. After page is full, other page is generated. The other page will have the same format with the first format. It will have page header followed by type header and field name of the types. In the first report, I assigned 4 byte for "TypeName". I changed its size as 16 bytes.

As it is shown in Table 2, page header has "PageID", "TotalSpace" and "FreeSpace" information. 116 bytes are reserved for future use. It is important to note that PageID describes how far away from the beginning of the file. If all the types is deleted in a page, page will still be allocated and its FreeSpace will be equal to TotalSpace minus header size which is equal to 900 Bytes. In other words, once the page is created, it will not be deleted any more.

As it is shown in Table 3, type header has "TypeID", "isUsed" and "NumberOfField" and "TypeName" information. One byte is reserved for future use. TypeID describes how far away from the beginning of the page. Even if the type is deleted, the space which is allocated for this type will not be deallocated. The program will just reset "isUsed" flag and reset all the other informations.

3.2 Data File Format

Data Files are also composed of the pages. Every page has header part and data part.

Table 4: Data File Format

Page Header	R#1 Header	R#1 F#1	R#1 F#2	...	R#1 F#10
R#2 Header	R#2 F#1	R#2 F#2	R#2 F#3	...	R#2 F#10
R#5 Header	R#5 F#1	R#5 F#2	R#5 F#3	...	R#5 F#10

Table 5: Page Header Format

PageID (4 byte)	TotalSpace(2 byte)	FreeSpace(2 byte)	Reserved(116 byte)

Table 6: Record Header Format

RecordID (1 byte)	isUsed(1 byte)	NumberOfField(1 byte)	Reserved(17 byte)

As it is shown in Table 4, Data Files start with a Page Header and follows by a record header and values of the its fields. Since every record has at most 10 fields; if a record is created, 10 x 4 + 20= 60 bytes allocated for this type even if it has less fields. After page is full, other page is generated. The other page will have the same format with the first format. It will have page header followed by record header and values of its fields. In the first report, I allocate 16 bytes for every field value. However, as it was stated in the conclusion part of the first report, it is too much for field value whose type is integer. 4 bytes will be enough to store field value information.

As it is shown in Table 5, page header has "PageID", "TotalSpace" and "FreeSpace" information. 116 bytes are reserved for future use. It is important to note that PageID describes how far away from the beginning of the file. If all the records of this page is deleted in a page, page will still be allocated and its FreeSpace will be equal to TotalSpace. In other words, once the page is created, it will not be deleted any more.

As it is shown in Table 6, record header has "RecordID", "isUsed" and "NumberOfField" information. 17 bytes are reserved for future use. RecordID describes how far away from the beginning of the page. Even if the record is deleted, the space which is allocated for this type will not be deallocated. The program will just reset "isUsed" flag and reset all the other informations.

4 Implementation Details

The pseudo codes which is given in detail in the first report was directly used to write the source codes. Therefore, it will be ambiguous to give all the pseudo codes again. Instead of giving the pseudo codes again, the classes will be introduced.

4.1 QueryManager Class

This class is written to listen the user's commands. Its object is created in the "main.cpp" function and enter a loop to get the user's command. User

can terminate the loop when it call "exit" command. The other commands are given below.

- Create a type
  " >> create table table_name column1_name column2_name ..."
- Delete a type
  " >> delete table table_name"
- List all types
  " >> list tables"
- Create a record
  " >> create record table_name field1_value field2_value ..."
- Delete a record
  " >> delete record table_name primary_key"
- Update a record
  " >> ¿¿ update record table_name field1_value field2_value ..."
- Search for a record via primary key
  " >> select record table primary_key"
- List all records of a type
  " >> list records table"

It is important to note that it is assumed that user will enter correct command all the time. Query Manager doesn't check user's input format. The other important note is that the commands are case sensitive.

4.2 PageManager Class

This class is responsible to manage reading and writing operations to file. When any information is needed to retrieve from the file, page manager "ReadPage" and "WritePage" methods are called. These methods will read entire page whose size is 1024 Bytes even if 1 byte is tried to be read. The instance will be created by passing file path as input to its constructor. After read operation, if it is realized that page is full, "WritePage" method should be called first. After that, "ReadPage" method can be called again.

4.3 StorageManager Class

StorageManager class includes all the DDL and DML operations. Singleton design pattern is used for this class. The instance will be instantiated once the GetInstance method is called and deallocated when the program terminates.

For the algorithms, the pseudo codes of the algorithms given in the first report is used exactly.

4.4 Container Classes

"Type", "Record", "Page" container classes is used to store the information neatly.Inside these classes, there are subclasses for their headers.

4.5 LogManager Classes

This class is used to organize the messages. Messages is categorized into four levels, namely, system messages, debug messages, error messages and info messages. System and error messages are active in default. However, whenever the user want to activate or deactivate these log, it can be done via sending a query.

- Activate Debug Messages
  " >> debug on"
- Deactivate Debug Messages
  " >> debug off"
- Activate Info Messages
  " >> info on"
- Deactivate Info Messages
  " >> info off"

5 Conclusion and Assessment

In this report, the implementation of Storage Manager was explained in detail. As it was stated, the fix length record length was used. Moreover, the records and types were stored in the system catalog and data files as heap files. Sorted file or hash file wasn't used.

The main advantage of using heap file is that it makes the programming and design easy.The other advantage is that it makes the delete and insert operation much more faster. However, there are many disadvantage to use heap file. It increases the select and update operations' complexity. In this design, the complexity of these operations directly proportional to the number of records or types in the storage. Nevertheless, it uses too much space even if it is not used because of the fact that after delete operation, we are not reorganizing the records or types after delete operation. Therefore, in this design, we can face with a scenario which page is allocated even if there is not any record inside it. Another disadvantage of this design is that it is not suitable the programs which use many threads which are trying to use our PageManager instance. The class is not thread-safe, therefore, it is very possible to observe race conditions.

There are two difference between the first report and second report. One of the is the size of the field value. As it was stated in the first report, I changed the size of the field value from 16 bytes to 4 bytes. The other difference is that I increased the size of "Typename" variable located in "Type Header".

Finally, I would like to remark that instead of designing command line interface, I designed query oriented program because most of the real world programs are working in this way. When I run "SQLite" or "MySql" program, it expects text commands from the user.