CmpE321 Introduction To Database Systems
Project 2
Storage Management System Implementation

İrem Uğuz
2015400165

April 7, 2019
2018-2019/Spring

Contents

1 Introduction To Project

1.1 Project Description

A database is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques. The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data.

In this project, I am expected to design a database management system(DBMS) that will support some basic DDL and DML operations. Also I will be designing the basic data structures like pages, records, files to make my storage management system fully functioning.

This report will be based on the previous project's report which is about the design of the system. In the differences with the previous project report will be highlighted in red in the assumptions and the constraints part. All of the storage structures have minor changes in them and their tables to please keep it in mind. The implementation of the storage manager is very similar to pseudo-code so that part is the same as first project report.

1.1.1 DDL Operations That Will Be Supported

1. Creating a type.
2. Deleting a type.
3. Listing all types.

1.1.2 DML Operations That Will Be Supported

1. Creating a record.
2. Deleting a record.
3. Searching for a record(by primary key).
4. Listing all records of a type
5. Updating a record

2 Assumptions And Constraints

There will be some basic assumptions and constraints to design the DBMS.

2.1 Assumptions

1. User will always enter valid input.
2. Field names are alphanumeric.
3. Type names are alphanumeric.
4. Fields are integers.
5. A disk manager already exists that is able to fetch the necessary pages when addressed.
6. The page size will be 512 Bytes.
7. The file size will be 5152 Bytes, meaning that a file can store 10 pages and a file header.
8. A page can only store the data about only one type.
9. A file can only store the pages with the same type of records.
10. A type can have at most 10 fields.
11. A type name can be at most 8 characters long(8 Bytes).
12. A type name must be at least 1 letters.(1 Bytes)
13. Type names must be unique. There can not be two different types with the same name.
14. A field name can be at most 8 characters long(8 Bytes).
15. A field name must be at least 1 letters.(1 Bytes)
16. Field names must be unique in a type. There can not be two different fields with the same name within a type.
17. User will not declare any key field while creating a type. The key field will be assumed to be the first field that the user gives.
18. User will give the number of fields that the given type will have when the type is created.
19. Records will be placed in pages in an atomic way. If a new record does not fit in the page, then that record will be placed in a new page.
20. Page header will contain number of records that are actively stored in the page,pointer to the first empty spot in the page and pointer to last used space in the page.
21. Record header will contain a boolean that indicates if the record is valid or deleted.

22. Because it is assumed that user gives valid input, the system won't control if the input is valid.
23. Hash function to determine which page that the record will be put in a file is the module of the key value of the record with 10.
24. File header will store the number of pages that is actively used in the file, pointer to the next and previous file of the same type, array that keeps the info if the usage conditions of the pages(if the page is used) and the array that keeps the info if the page is full.
25. User won't try to delete a non existing record.
26. Size of the records of each type will be fixed.
27. User won't try to add a record whose type does not exist.
28. User won't try to add a record whose type does not exist. Also user can not try to do any operation except the search operation for a record whose type doesn't exist.
29. The system will delete the files of the deleted records if there is no record left of that type.
30. There won't be any .txt files with numeric names in the path that the program works as those files will be created and managed by the system.
31. The user can not create a SystemCatalog.txt file in the directory of the program.

2.2 Constraints

1. Data will be organized in pages.
2. Pages must contain records.
3. All pages can not be stored in a single file.
4. A file must contain multiple pages.
5. System must create new files as the storage manager grows.
6. If files become empty after deletions, the system should delete those files.
7. System must not load the whole file to memory when needed.
8. Instead system must read files page by page.
9. While types will be listed by ascending type names, records will be listed by ascending primary key.
10. Search, update and delete of records will always be done by primary key.
11. When a type is deleted, all records of that type must be deleted.

3 Storage Structures

3.1 System Catalog

System catalog is where the information about types and their fields are kept in the system. The system catalog file must exist so the data base system can work properly. Also the system does not allow the system catalog file to be deleted and any new catalog to be created.

System Catalog Header			
Number of types in the database			
Type Name	**# of Fields**	**FieldNames[]**	**File Id**
Type 1	5	Field Names[]	Id1
Type 2	7	Field Names[]	Id2

System catalog file's header stores the number of types. The system catalog file contains the field names of each type, field number of each type, type name and id of the first file that is associated with that type. If there is no file opened yet for that type or the type doesn't contain any record, the id will be -1 until an entry is made. The field names are put in an array and there is 10 fields in array. If there is less than 10 fields, empty strings are kept in the array.

3.2 Data File

Data file is basically files that contains pages. A file can store data only about a single type so its pages must contain only one type of record.

Data file's header contains number of pages that are used in the file, id of the next file about that type if exists,id of the previous file of the type, an array that keeps the info about if the pages are used and another array about the info of if the page is full. The id of the previous file will be -1 if the file is the first file about that type. The file will be containing 10 pages in it.

Each file will be containing 10 pages and a header section. In header section, number of pages used will be an integer, 4 bytes;type name can be 8 bytes, previous and next page ids will be integers in total of 8 bytes and the boolean arrays will take 20 bytes. The whole size of the header will be 32 bytes. Since the file will contain 10 pages, each 512 byte, 5120 bytes total; the assigned size of the file will be 5152 bytes.

Data File Header									
# of Pages Used									
Pointer To Next File									
Pointer To Previous File									
The Page Used Boolean Array									
0	1	2	3	4	5	6	7	8	9
The Page Full Boolean Array									
0	1	2	3	4	5	6	7	8	9
Page#0		Page#1		Page#2		Page#3		Page#4	
Page#5		Page#6		Page#7		Page#8		Page#9	

3.3 Data Page

Page is where the records will be stored. A page can store only one type of record. Page header will contain a pointer to the first empty slot in the page, the pointer to the last used space in the page and the total number of records that are stored in the page.

When records are stored in a page, they will always be put to the top place that is empty. Also, when a record is put in a page, if the record doesn't fit to the empty space in the page, it will need to be put in another page.

Page Header
Pointer To The First Empty Slot
Pointer To End Of Used Space
# of Records
Record#0
Record#1
Record#2

Page headers size will be 12 bytes pointers and number of the records is 4 bytes each. And because a page has a fixed size of 512 bytes, a page will contain minimum of 12 records because maximum size of a record is 41 bytes.

3.4 Record

Records are the basic storage units that we store data. A record has the field values of its associated type. Field values are integers as stated in the assumptions. A record's first fields is its key field and this fields must be unique among all the records belonging to a certain type. This key will be the main point that we store the records as I

will use the hashing method. Record header will contain a boolean to represent if the record is valid or not(meaning deleted).

Record Header						
Valid Boolean						
Field#0	Field#1	Field#2	Field#3	Field#4	Field#5	Field#6
15	7	21	8	256	420	63

A record can be at most 41 bytes as each field is an integer, 4 bytes and at most 10 field and the header is 1 byte; 1 bytes for valid boolean. Of course, as the number of fields in a record can vary, this is the maximum size of the record as a record can have at most 10 fields.

4 Operations

4.1 Creating A Type

Algorithm 1 createType()

```
1: typeName = getInput()
2: fieldNum = getInput()
3: if systemCatalog.typeExists(typeName) then
4:     ErrorMessage(Type already Exists)
5: end if
6: fields[fieldNum]
7: for i from 0 to fieldNum do
8:     fields[i] = getInput()
9: end for
10: keyField = getInput()
11: takeArrayElementToFront(fields,keyField)
12: updateSystemCatalogHeader(getSystemCatalogHeader()+1)
    {//Types in system catalog can be seen here3.1}
13: newType = new Type(typeName,fieldNum,fields,keyField, null)
14: addNewTypeToSystemCatalog(newType)
15: return true
```

4.2 Deleting A Type

Algorithm 2 deleteType()

```
1: typeToDelete= getInput()
2: dType = findInSystemCatalog(typeToDelete)
   {//Data file can be seen here3.2}
3: firstFile = FileManager.getFileHeader(dType.filePointer)
4: if file == null then
5:    removeFromSystemCatalog(dType)
6: end if
7: pageArray = file.pageArray
8: for i from 0 to 9 do
9:    deletePage(pageArray[i])
10: end for
11: while file.nextFile != null do
12:    prev = file
13:    file = file.nextFile
14:    prev.nextFile = null
15:    pageArray = file.pageArray
16:    for i from 0 to 9 do
17:       deletePage(pageArray[i])
18:       pageArray[i]=null
19:    end for
20: end while
```

Algorithm 3 deletePage()

```
   {//Contents of page can be seen here3.3} recordPointer =
   page.pointerToFirstRecord for i from 0 to page.numOfRecords do

3: deleteRecord(recordPointer)
4: recordPointer++
5: end for
```

4.3 Listing All Types

Algorithm 4 listAllTypes()

```
1: numOfTypes = systemCatalog.getNumOfTypes()
2: for i from 0 to numOfTypes do
3:    ttype = systemCatalog.getType(i)
4:    print(ttype)
5: end for
```

4.4 Creating A Record

Algorithm 5 createRecord()

```
1: recordType= getInput()
2: type = systemCatalog.findInSystemCatalog(recordType)
3: numOfFields = type.numOfFields
4: fields[numOfFields]
5: for i from 0 to numOfFields do
6:    fields[i] = getInput()
7: end for
   {//Record can be seen here3.4}
8: record = new Record(recordType, fieldArray, numOfFields)
9: dataFile = type.filePointer()
10: if dataFile == null then
11:    type.filePointer = createDataFile(null,recordType)
12: end if
13: pageNum = hashFunc(typeName, record.fields[0])
14: page = dataFile.pageArray[pageNum]
15: location
16: if page == null  then
17:    dataFile.pageArray[pageNum] = createPage(pageNum)
18:    page = dataFile.pageArray[pageNum]
19:    location = page.addRecord(record)
20: else
21:    if sizeOf(record)+pointerToEmptySlot ¡= 512KB then
22:       location = page.addRecord(record)
23:    else
24:       newFile = createDataFile(dataFile, typeName)
25:       dataFile.nextFile = newFile
26:       newFile.pageArray[pageNum] = createPage(pageNum)
27:       page = newFile.pageArray[pageNum]
28:       location = page.addRecord(record)
29:    end if
30: end if
31: page.nextEmptySlot = location+sizeOf(record)
```

Algorithm 6 createDataFile(previousFilePointer,typeName)

```
1: file = FileManager.createFileWithSize(5KB)
   {//File header has the fields nofPagesUsed, typeName, pointerToNExtFile,
   pointerToPrevFile, pointerToPageArray as stated here 3.2}
2: file.addHeader(0,typeName, null, previousFilePointer, null)
3: return file
```

Algorithm 7 createPage(pageId)

```
1: page = new Page(pageId,getPointerToBeginning(),getPointerToBeginning(),0)

2: return page
   {//Page structure can be seen here3.3}
```

Algorithm 8 page.addRecord(record)

```
1: page.numOfRecords++
2: page.put(page.nextEmptySlot,record)
3: nextEmpty = page.nextEmptySpot
4: loc = page.firstRecord
5: while !isEmpty(loc + sizeof(record)) do
6:    loc =+ sizeof(record)
7: end while
8: page.nextEmptySlot = loc
9: return nextEmpty
```

4.5 Deleting A Record

Algorithm 9 deleteRecord()

```
1: primaryKey = getInput()
2: type = getInput()
3: file = systemCatalog.findType(type).getFilePointer()
4: hashValue = hashFunc(type, primaryKey)
5: page = file.pageArray[hashValue]
6: while page != null and file.nextFile!=null do
7:    loc= page.getFirstRecordPointer
8:    found = false
9:    for i from 0 to page.numOfRecords do
10:      if page.getRecordAt(loc).fields[0]==primaryKey then
11:         found = true
12:         page.getRecordAt(loc).validBit(0)
13:         page.nextEmptySlot = loc
14:         page.numOfRecords–
15:         if page.numOfRecords == 0 then
16:            file.pageArray[hashValue] = null
17:            file.nofPagesUsed –
18:            if file.nofPagesUsed == 0 then
19:               prev = file.previousFile
20:               if prev == null then
21:                  systemCatalog.findType(type).filePointer = null
22:               else
23:                  next = file.pointerToNextFile
24:                  prev.pointerToNextFile = next
25:               end if
26:            end if
27:         end if
28:         break
29:      end if
30:   end for
31:   if found then
32:      break
33:   end if
34:   file = file.nextFile
35:   page = file.pageArray[hashValue]
36: end while
```

4.6 Searching For A Record

Algorithm 10 findRecord()

```
1:  primaryKey = getInput()
2:  type = getInput()
3:  file = systemCatalog.findType(type).getFilePointer()
4:  hashValue = hashFunc(type, primaryKey)
5:  page = file.pageArray[hashValue]
6:  record = null
7:  while page != null and file.nextFile!=null do
8:     loc= page.getFirstRecordPointer
9:     found = false
10:    for i from 0 to page.numOfRecords do
11:       if page.getRecordAt(loc).fields[0]==primaryKey then
12:          found = true
13:          record = page.getRecordAt(loc)
14:          break
15:       end if
16:    end for
17:    if found then
18:       break
19:    end if
20:    file = file.nextFile
21:    page = file.pageArray[hashValue]
22: end while
23: return record
```

4.7 Listing All Records Of A Type

Algorithm 11 listRecords(type)

```
1: file = systemCatalog.findType(type).getFilePointer()
2: if file == null then
3:    print(No record for this type.)
4: end if
5: while file != null do
6:    for i from 0 to 10 do
7:       if file.pageArray[i] != null then
8:          page = file.pageArray[i]
9:          pointer = page.firstRecordPointer
10:         for i from 0 to page.nofRecords do
11:            print(page.getRecordAt(pointer))
12:            pointer += sizeOf(page.getRecordAt(pointer))
13:         end for
14:      end if
15:   end for
16:   file = file.nextFile
17: end while
```

5 Conclusions And Assessment

For this project, I had to design a data base management system and saw that it can be a very confusing task to do. At first I needed to understand the data base storage system very well. Even if I thought, I understood the main idea, as I progressed to write the pseudo code, I realized that I didn't understand everything clearly. I had to change my data file, page and record as I wrote the pseudo code because I realized that I added unnecessary elements to the headers. In that aspect, I believe that writing a pseudo code helped me understand the database concept better. Because, without thinking about implementation, the ideas were too abstract in my head.

When designing this data base, I decided to use hashing method to make the search and listing faster. But as I thought there could be a lot of types and lots of pages, I am not sure if my hashing will make lots of difference on the speed or it could be better to just add the records linearly to the pages because I have 10 pages for hashing and as the size of the data will grow, the overflow of the pages will increase. But I am hoping that with a good hashing method, this algorithm will be faster.

Because of my concerns about the number of the types, I gave the pages a smaller number in case there are too many types. But as the expectation of the size of the data will grow, the data size that is determined to be given to pages and the files can grow too. In case of an expectation of growth, the number of pages in a file can increase and the necessary change of hash function can be made to modify the function to result to be between larger numbers. For now, I thought it would be practical to keep the size of the pages and files small, in case there is fewer records for each type.