CmpE321 Introduction To Database Systems
Project 1
Storage Management System Design

İrem Uğuz
2015400165

March 10, 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.

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

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 5150 Bytes, meaning that a file can store up to 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 10 characters long(10 Bytes).
12. A type name must be at least 3 letters.(3 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 10 characters long(10 Bytes).
15. A field name must be at least 3 letters.(3 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 declare the key field when a type is created. While declaring, it doesn't have to be the first field that is given but when storing, key field will be the 0th element of the field array.
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 this id, number of records that are actively stored in the page,pointer to the first record that is stored in the page and pointer to next empty slot in the page that a record can be inserted.
21. Record header will contain the type name, number of fields and the pointer to the field array.
22. Because it is assumed that user gives valid input, the system won't control if the input is valid.
23. It is assumed that the hash function used for each type is able to give good hash values that won't cause too much overflow in the pages. Meaning that there won't be too many records that gives the same result. The results will be divided nearly equally to results. The hash function's results are integer numbers between 0-9.

24. File header will store the number of pages that is actively used in the file, type name of the file, pointer to the next and previous file of the same type and pointer to the page array.

25. User won't try to delete a non existing record.

26. Size of the records of each type will be fixed.

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.

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[]**	**Primary Key**	**Pointer**
Type 1	5	Field Names[5]	Name	Pointer1
Type 2	7	Field Names[7]	Year	Pointer2

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, primary key of each type and a pointer to 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 pointer will be null until an entry is made.

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 the info about the type, number of pages that are used in the file, pointer to the next file about that type if exists, pointer to the array that stores the pages and pointer to the previous file about that type. The pointer about the previous file will be null if the file is the first file about that type. The file will be containing 10 pages in it. Pointers to the pages in the file will be stored in an array in the file as well.

Data File Header				
# of Pages Used				
Type Name				
Pointer To Next File				
Pointer To Previous File				
Pointer To The Page 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

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 10 bytes, pointers will take 12 byte as a pointer size is 4 byte. The whole size of the header will be 26 bytes. Since the file will contain 10 pages, each 512 byte, 5120 bytes total; the assigned size of the file will be 5150 bytes, giving 4 bytes to avoid some error.

3.3 Data Page

Page is where the records will be stored. A page can store only one type of record. Each page will have its id. Page header will contain this id, number of records that are actively stored in the page,pointer to the first record that is stored in the page and pointer to next empty slot in the page that a record can be inserted.

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
Page ID
Pointer To The First Record
Pointer To Next Empty Slot
# of Records
Record#0
Record#1
Record#2

Page headers size will be 16 bytes as its id, 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 8 records because maximum size of a record is 58 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 the type name, number of fields and the pointer to the field array.

Record Header						
Record Type						
Pointer To The Field Array						
# of Fields						
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 58 bytes as each field is an integer, 4 bytes and the header is 18 bytes; 10 bytes for type name, and 4 bytes for pointer and number of fields. 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()

```
typeName = getInput()
fieldNum = getInput()
if systemCatalog.typeExists(typeName) then
  ErrorMessage(Type already Exists)
end if
fields[fieldNum]
for i from 0 to fieldNum do
  fields[i] = getInput()
end for
keyField = getInput()
takeArrayElementToFront(fields,keyField)
updateSystemCatalogHeader(getSystemCatalogHeader()+1)
{//Types in system catalog can be seen here3.1}
newType = new Type(typeName,fieldNum,fields,keyField, null)
addNewTypeToSystemCatalog(newType)
return true
```

4.2 Deleting A Type

Algorithm 2 deleteType()

```
typeToDelete= getInput()
dType = findInSystemCatalog(typeToDelete)
{//Data file can be seen here3.2}
firstFile = FileManager.getFileHeader(dType.filePointer)
if file == null then
  removeFromSystemCatalog(dType)
end if
pageArray = file.pageArray
for i from 0 to 9 do
  deletePage(pageArray[i])
end for
while file.nextFile != null do
  prev = file
  file = file.nextFile
  prev.nextFile = null
  pageArray = file.pageArray
  for i from 0 to 9 do
    deletePage(pageArray[i])
    pageArray[i]=null
  end for
end while
```

Algorithm 3 deletePage()

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

  deleteRecord(recordPointer)
  recordPointer++
end for
```

4.3 Listing All Types

Algorithm 4 listAllTypes()

```
numOfTypes = systemCatalog.getNumOfTypes()
for i from 0 to numOfTypes do
  ttype = systemCatalog.getType(i)
  print(ttype)
end for
```

4.4 Creating A Record

Algorithm 5 createRecord()

```
recordType= getInput()
type = systemCatalog.findInSystemCatalog(recordType)
numOfFields = type.numOfFields
fields[numOfFields]
for i from 0 to numOfFields do
   fields[i] = getInput()
end for
{//Record can be seen here3.4}
record = new Record(recordType, fieldArray, numOfFields)
dataFile = type.filePointer()
if dataFile == null then
   type.filePointer = createDataFile(null,recordType)
end if
pageNum = hashFunc(typeName, record.fields[0])
page = dataFile.pageArray[pageNum]
location
if page == null  then
   dataFile.pageArray[pageNum] = createPage(pageNum)
   page = dataFile.pageArray[pageNum]
   location = page.addRecord(record)
else
   if sizeOf(record)+pointerToEmptySlot ¡= 512KB then
      location = page.addRecord(record)
   else
      newFile = createDataFile(dataFile, typeName)
      dataFile.nextFile = newFile
      newFile.pageArray[pageNum] = createPage(pageNum)
      page = newFile.pageArray[pageNum]
      location = page.addRecord(record)
   end if
end if
page.nextEmptySlot = location+sizeOf(record)
```

Algorithm 6 createDataFile(previousFilePointer,typeName)

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

Algorithm 7 createPage(pageId)

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

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

Algorithm 8 page.addRecord(record)

```
page.numOfRecords++
page.put(page.nextEmptySlot,record)
nextEmpty = page.nextEmptySpot
loc = page.firstRecord
while !isEmpty(loc + sizeof(record)) do
   loc =+ sizeof(record)
end while
page.nextEmptySlot = loc
return nextEmpty
```

4.5 Deleting A Record

Algorithm 9 deleteRecord()

```
primaryKey = getInput()
type = getInput()
file = systemCatalog.findType(type).getFilePointer()
hashValue = hashFunc(type, primaryKey)
page = file.pageArray[hashValue]
while page != null and file.nextFile!=null do
  loc= page.getFirstRecordPointer
  found = false
  for i from 0 to page.numOfRecords do
    if page.getRecordAt(loc).fields[0]==primaryKey then
      found = true
      page.getRecordAt(loc).validBit(0)
      page.nextEmptySlot = loc
      page.numOfRecords–
      if page.numOfRecords == 0 then
        file.pageArray[hashValue] = null
        file.nofPagesUsed –
        if file.nofPagesUsed == 0 then
          prev = file.previousFile
          if prev == null then
            systemCatalog.findType(type).filePointer = null
          else
            next = file.pointerToNextFile
            prev.pointerToNextFile = next
          end if
        end if
      end if
      break
    end if
  end for
  if found then
    break
  end if
  file = file.nextFile
  page = file.pageArray[hashValue]
end while
```

4.6 Searching For A Record

Algorithm 10 findRecord()

```
primaryKey = getInput()
type = getInput()
file = systemCatalog.findType(type).getFilePointer()
hashValue = hashFunc(type, primaryKey)
page = file.pageArray[hashValue]
record = null
while page != null and file.nextFile!=null do
    loc= page.getFirstRecordPointer
    found = false
    for i from 0 to page.numOfRecords do
        if page.getRecordAt(loc).fields[0]==primaryKey then
            found = true
            record = page.getRecordAt(loc)
            break
        end if
    end for
    if found then
        break
    end if
    file = file.nextFile
    page = file.pageArray[hashValue]
end while
return record
```

4.7 Listing All Records Of A Type

Algorithm 11 listRecords(type)

```
file = systemCatalog.findType(type).getFilePointer()
if file == null then
  print(No record for this type.)
end if
while file != null do
  for i from 0 to 10 do
    if file.pageArray[i] != null then
      page = file.pageArray[i]
      pointer = page.firstRecordPointer
      for i from 0 to page.nofRecords do
        print(page.getRecordAt(pointer))
        pointer += sizeOf(page.getRecordAt(pointer))
      end for
    end if
  end for
  file = file.nextFile
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.