Cmpe321 - Introduction to Database Systems

Spring 2020

Project 1- DBMS Design

Afra Akbaş
2015400024

Contents

1 Introduction

In this project, I will design a storage management system. It has a system catalog for storing the metadata and data storage units (files, pages, and records) for storing the actual data. It supports the followings:

- DDL operations(Create a type, delete a type, and list all types)
- DML operations(Create a record, delete a record, search a record, update a record, and list all records of a type)

To design a storage management system, I followed the procedure below. I determined my constraints and assumed some features already exist. According to these constraints and assumptions, I designed my storage structures and operations. I described my structures with tables and diagrams to make them more understandable. Not only structures but also I explained operations and the algorithms behind the operations, supported with pseudo-codes. At the end, I explained my system with pros and cons.

2 Assumptions & Constraints

While designing my system, I made some assumption. Some of the assumptions are given in the project description.

2.1 Assumptions

- User always enters valid input.
- All fields shall be integers. However, type and field names shall be alphanumeric.
- A disk manager already exists that is able to fetch the necessary pages when addressed.
- Page size is 1.2 KB.
- File size is 25 KB.
- A page contains at most 25 records.
- A file contains at most 20 pages.
- A page header contains the page id, the pointer of the next page, type of a page, and the number of empty records.
- A record header contains the record id, the pointer of the next record, and information about emptiness of the record.
- A page contains only one type of record.

- Pages in a file can be pages for different types of records.
- Max number of fields a type can have is 10.
- Max length of a type name 15.
- Max length of a field name 15.

There are also constraints for our projects.

2.2 Constraints

- The data must be organized in pages, and pages must contain records.
- All pages cannot store in the same file, and a file must contain multiple pages. This means that the system must be able to create new files as storage manager grows. Moreover, when a file becomes free due to deletions, that file must be deleted.
- 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.
- 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.

3 Storage Structures

We should design the structures of the storage system. These structures consist of pages, files, records, and a system catalog. Their purpose is to keep the information in order and making them more accessible.

3.1 System catalogue

The system catalog is a file named sys_cat.cat. It stores the metadata of the system. It contains a list of types that includes types of records, the number of fields of this type, and the names of these fields. When a new type created, it is added into the system catalog with related entries(name, fields, etc.). To delete a type, we remove it from the system catalog. In addition to that, it also has a list of files that the system has.

Type #1						File #1
Type #2						File #2
Type Name #3	# of fields	name-1	name-2	...	name-10	File #3
...						...

3.2 File Design

The file size in this system is 25 KB. One file contains at most 20 pages. It contains a file header(id, pointer, etc.) and pages.

The file id: 4 bytes
The pointer of next file: 4 bytes
The number of empty pages: 1 bytes

Header
Page #1
Page #2
...
Page #20

3.3 Page Design

The page size in the system is 1.2 KB. One page contains at most 25 records. For each type of record, there must be at least one page because each page contains the same-type records. A page header consists of the page id, the pointer of the next page, the number of empty record, and the page type.

The page id: 4 bytes
The pointer of next pointer: 4 bytes
The page type: 4 bytes
The number of empty records: 1 bytes

Header	record #1	record #2	...	...

3.4 Record Design

A record has a header and fields which can be at most 10. The header has information about the record. The record id, the pointer of the next record, a flag for emptiness. The record id is unique for all records, which means it is the primary key.

The record id: 4 bytes
The pointer of next record: 4 bytes
isRegistered: 1 bytes

Header	Field #1	Field #2	...	...	Field #10

4 Operations

4.1 DDL operations

Pseudo-code for DDL operations

4.1.1 Create a type

This algorithm creates a type and adds it to the database. It fills the name of the type, number of fields, and names of them. Then, it creates a page for this new type and adds it to the file if there is a space. Otherwise, it also creates a file and adds the page to this new file. In the end, it adds the new type and the new file to the system catalog.

Algorithm 1 Create a type

Input: $typeName, fieldNumber, fieldNames$

```
open(sys_cat.cat)
nameOfType ← typeName
numberOfFields ← fieldNumber
for i = 0 ← fieldNumber − 1 do
    nameOfFields[i] ← fieldNames[i]
end for
newPage ← createPage(typeName)
for f ∈ files do
    if f.numberOfEmptyPages > 0 then
        f.add(newPage)
        add type into sys_cat.cat
        return
    end if
end for
newFile ← createFile(newPage)
add the type and the newFile into sys_cat.cat
close(sys_cat.cat)
```

4.1.2 Delete a type

This algorithm deletes a type from the database. It takes the type list from the system catalog and finds the type with a given name. Then, deletes all attributes and pages related to this type. It also removes this type from the system catalog.

Algorithm 2 Delete a type

Input: *typeName*

```
open(sys_cat.cat)
for t ∈ types do
  if t.typeName = typeName then
    delete all attributes
    break
  end if
end for
delete all pages with typeName
close(sys_cat.cat)
```

4.1.3 List all types

This algorithm lists all types in the database. It takes the type list from the system catalog and lists them.

Algorithm 3 List all types

Output: *types*

```
open(sys_cat.cat)
for t ∈ types do
  print(t)
end for
close(sys_cat.cat)
```

4.2 DML operations

Pseudo-codes for DML operations:

4.2.1 Create a record

It creates a new record according to the given type. It fills the id of the record, related fields, and the flag of the record. Then, finds the pages of this type. If there is a space on a page, adds the record into the page. Otherwise, it creates a new page with the given type name and tries to find a space in files. If there is a space in a file, adds the page into this file. If there is no space, it creates a new file and adds a new page into this new file.

Algorithm 4 Create a record

Input: $typeName, id, fieldNumber, fieldValues$

```
open(sys_cat.cat)
recordID = id
for i = 0 ← fieldNumber − 1 do
  fields[i] ← fieldValues[i]
end for
isRegistered = TRUE
pages ← findPages(files, typeName)
for p ∈ pages do
  if p.numberOfEmptyRecords > 0 then
    p.add(this.record)
    return
  end if
end for
newPage ← createPage(typeName, this.record)
for f ∈ files do
  if f.numberOfEmptyPages > 0 then
    f.add(newPage)
    return
  end if
end for
newFile ← createFile(newPage)
add the newFile into sys_cat.cat
close(sys_cat.cat)
```

4.2.2 Delete a record

It deletes the given-type-record from the associated page. It finds pages with the given type name. Then, finds the records on these pages. It deletes all attributes of this record and changes its flag with empty.

Algorithm 5 Delete a record

Input: $typeName, recordId$

$open(sys\_cat.cat)$
$pages \leftarrow findPages(files, typeName)$
$r \leftarrow findRecord(pages, recordId)$
$r.isRegistered = FALSE$
delete related fieldValues
$close(sys\_cat.cat)$

4.2.3 Search a record

It searches for a record by a primary key. It takes the primary key of the record and the type name. It finds the pages with the given type name and checks the id of records on the pages, whether it equals the primary key or not. If it does, then it returns the record.

Algorithm 6 Search a record

Input: $typeName, primaryKey$
Output: $record$

$open(sys\_cat.cat)$
$pages \leftarrow findPages(files, typeName)$
for $p \in pages$ **do**
  for $r \in p.records$ **do**
    if $r.recordId = primaryKey$ **then**
      return r
    end if
  end for
end for
$close(sys\_cat.cat)$

4.2.4 Update a record

It updates the fields of a record by a primary key. It takes the primary key of the record and the type name. It finds the pages with the given type name, then records with the given primary key. It checks whether it equals the primary key or not. If it does, then it updates the fields of the record.

Algorithm 7 Update a record

Input: $typeName, primaryKey, updatedFields$

```
open(sys_cat.cat)
pages ← findPages(files, typeName)
r ← findRecord(pages, primaryKey)
r.updateFields(updatedFields)
close(sys_cat.cat)
```

4.2.5 List all records of a type

It lists all records in the database for a given type. It finds the same-type-pages with the given type name. Then, it lists all the records on the page.

Algorithm 8 List all records of a type

Input: $typeName$
Output: $records$

```
open(sys_cat.cat)
pages ← findPages(files, typeName)
for p ∈ pages do
  for r ∈ p.records do
    print(r)
  end for
end for
close(sys_cat.cat)
```

5 Conclusion & Assessments

In conclusion, I designed a simple database storage system. I tried to make my system more accessible and feasible in many ways. However, of course, it has many pros and cons.
One of the advantages of my database system is that it is easy to access the same types of records since each page has a type-name. A page can only have one type of record. Therefore, it is like clustering records into pages. It is also an advantage for the other way around. I designed my system in a way that you can fetch a record from a different page of the same file. It means that it is also easy to access different types of records because a file contains pages of various type-name.
On the other hand, there are many disadvantages that this property can cause. All same-type-pages spread over all files. They do not cluster into files, so it makes it hard to find them.