Cmpe321 - Spring2021- HALO project

Berkay Demirtaş
2017400234
Berkant Koç
2020400342
Melih Özcan
2017400165

July 4, 2021

Contents

1 Introduction

In this project, we were asked to design and implement a database system for a very important mission. Operations should be handled are definition language operations, authentication language operations and management language operations. For detailed operation list you can check project description. There is only one page space in the main memory. Therefore we need to implement operations by taking page size pieces from files, updating them and writing them back. Also we need to write a line to log file after each operation.

Database storage design is an important concept. Designing files, pages, and records and their headers is a crucial topic. Clever design of these stuff can increase our efficiency and can reduce our burden when dealing with input/output operations related to databases. In this Project, we first planned how to represent files and pages (txt file etc.) then decided how many pages should a file contain and how many bytes should be a page. After that, we determined what file, page, and record headers will stand for. After doing this process, we put the records inside the pages and files according to how we design the storage mechanism. Then, according to the inputs, we fetched the records from pages (by doing some byte manipulations) and did the asked input operations and prompted outputs to the output file. Another crucial concept is creating a Log File. We can use a log file to recover from application or system errors in case of a crash. Therefore, logging all the authentication, definition, and management operations into the log file is necessary. So, we created a csv file holding all those operations. The csv file contains four column: username, occurrence, operation, and status; we stored the informations of finished operations here.

2 Assumptions & Constraints

Clearly specify your assumptions and constraints of the system in an itemized or tabular format.

2.1 Assumptions

- Spaces at the end of the any kind of input such as field names and field values are ignored. For example, for our system "aaa" and "aaa " are the same.
- In filter record command , we assumed that condition doesn't contains spaces. For example "a < b" is invalid for our system.
- Our time values in log file is double, not int to see the time line.

2.2 Constraints

1. We stored space instead of "-" in $E226 - S187$. However we handle it in during the I/O operations. Therefore users can't realize this difference.

2. Maximum number of field for each type is 14 and length of them is 20 bytes at most
3. Length of the field values are at most 20 bytes.

3 Storage Structures

Below figure explains our file and page structure. In each file there is a file header and 5 pages. File header is 312 bytes and each page is 2 Kb . We store records that have different types in different files. Therefore in the first 20 bytes of the file header, we have typeName information. (Our storage structures are fixed size and whenever there are more than enough bytes for a information, we fill unnecesary bytes with blank symbols. For example if our type name is ”human” and there are 20 spaces to store this information, we store blank symbol in last 15 bytes.) In the preceding 2 bytes of file header, we store number of fields of type. After that , we store 20 bytes for each field name and number of field names might be at most 14. (Total 14*20 bytes). As I explained above, for unnecessary places, white space is used. In the last 10 bytes of the file header, there is information about pages in our file. We store 1 byte to check if page is full and one byte to check if page is empty.

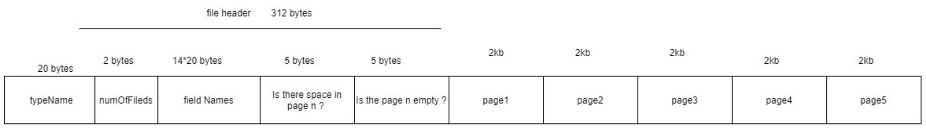

Figure 1: file structure

Below figure shows page structure. Each page can store a page header and 7 records. Page header stores 1 byte that represents if the record n is empty or not for each record. Size of the records are 20*14 bytes because each record has at most 14 fields and value of each field can be at most 20. Also at the end of the pages, there are some extra spaces to round up page size 2 Kb.

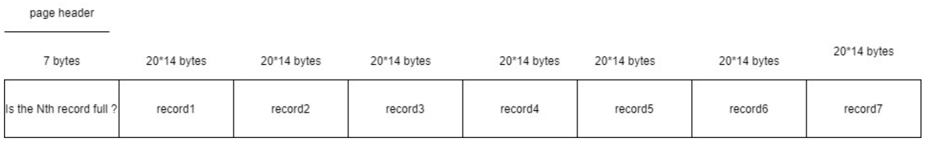

Figure 2: file structure

As system catalog, we store user names, passwords, type names, number of records in total and number of records for each type.

For each type, we are creating a new file with the name (typeName + 1 + .txt). When the first file is full, we create next file by changing 1.

4 Operations

4.1 HALO Authentication Language Operations

1. Register : We are handling this operation in the main

   ```
   take username, password1, password2
   if password1 == password2 and if username is not in Users(userName.txt)
   then create new user
   else fail
   ```

2. Login : We are handling this operation in the main

   ```
   take username, password
   if userDict[username] == password
   then login and user = username
   else fail
   ```

3. Logout : We are handling this operation in main

   ```
   if user != null
   then user = null
   else fail
   ```

4.2 HALO Definition Language Operations

1. Create Type: We are handling this operation in createTypeFile method

   ```
   if typeName exists:
   return 0
   else
   create a txt file with name (typeName + fileNum + .txt)
   write default file and page headers by using createNspaceString and createFieldNamesString methods.
   return 1
   ```

2. Delete Type: We are handling this operation in deleteType method

   ```
   if typeName doesn't exists
   return 0
   else
   find last file of the type with findLastTypeFile method
   delete all preceding files
   ```

3. Inherit type: We are handling this operation in inheritType method.

   ```
   create path of sourceFile and targetFile
   ```

```
if path exists for sourceFile and doesn't exists for targetFile
get header of source file
parse field names with parseFieldNamesFromHeader method
append them with additional fieldNames
call createTypeFile method
```

4. List type: We are handling this operation in inheritType method

```
for each typeName in typeNames(typeNames.txt)
counter++
print to output file
if counter == 0
return 0
else return 1
```

4.3 HALO Management Language Operations

1. Create record: we are handling this operation in createRecord method. This is a complex method, therefore I can't explain it as pseuducode. First, we find the last file for given typeName. After that we insert the record to the first page with the space and change page and file header accordingly. To preserve ordering, we swap last inserted record with the neighbour records recursively until it is in the correct file and page. This swapping operation done by swapTwoRecord method which takes file, page and record number of 2 records and swap them.

2. Delete record: we are handling this operation in deleteRecord method. This is a complex method, therefore I can't explain it as pseuducode. First, we find the record to delete with same algorithm I will explain in Search Record section. After that we swap it with the neighbour records by using swapTwoRecords2 to put it end of the last file. When our record is in the end of the file, we replace it with space characters and changing page and file headers accordingly by calling deleteLastRecordOfType method.

3. Update record: we are handling this operation in deleteRecord method.

```
find last file by calling findLastFileForType
for each file
  for each page
    for each record
      if 2. field == primaryKey :
        create a new record and write it on the old one
        counter++
      else continue
if counter == 0
```

```
    return 0
else
    return 1
```

4. Search record: we are handling this operation in searchRecord method.

   Same algorithm with update record but just printing the found record not updating.

5. List record: we are handling this operation in searchRecord method.

   Same algorithm with update record but just printing the all records without checking any conditions.

6. Filter record :we are handling this operation in filterRecord method.

   Parameters of this method is important to understand.So for this method, I will also explain what the parameters are. LeftIndex and RightIndex are left and right part of the condition. They might be a number or name of a field. typeOfCondition2 is comparision sign like ==, <= etc. . The values that typeOfCondition can take is 0,1,2,3 for number-number , field-num, num-field, field-field in order. In the method, we traverse all records of given type with the same algorithm with searchRecord method. Then we find the values of index1 and index2 for that record if they are field names. We are concataneting this values with typeOfCondition2 as a string and finding boolean result with eval() method in python. If the result of the boolean true, then we print that record and increase counter by one. At the end, counter shows us if any result have been found and decides the return value.

5 R & D Discussions

Now our system can only support one user.If we want to increase number of users that use system at the same time, we can shared-nothing architecture to paralelize individual operations and make querries faster. If we encounter any crash during the querry evaluation, we might use 2PC protocol to undone some changes during restart after failure. Distributed operations may require system to store a record more than one which is a big overhead.

6 Conclusion & Assessment

It seems like our project is working fine and thanks to the fixed siz structure of our project, we can easily implement new operations and features for our database. However this design is not resource efficient because for example to store a field name which has 2 bytes length, we need to store 20 bytes. Therefore

18 bytes are useless. Because of this, too many new file might be needed in to store a large amount of data. When the number of records stored in each file decreases, we need to store page headers and file headers more. They are also consume resources.