CMPE 321

Project 2

SUMMER TERM 2019

A Simple Storage Manager System Design

2018400279
Alperen Bağ

July 30, 2019

Contents

1 Introduction

In this project, I designed and implemented a small-scale database with fundamental techniques. My design is a conceptual design, which means it is not appropriate for the practical use. Also I made the design with some assumptions which are stated in the following section. The implementation of this design is able to store multiple types and each type can have maximum 1600 instances.

2 Assumptions & Constraints

2.1 Assumptions

- There shall be no error checking mechanism, because valid inputs are guaranteed.
- Type and field names shall be constituted of alphanumeric characters.
- An integer is 4 bytes and a char is 1 bytes (ASCII encoding)
- There shall not be more than one type that have the same type name.

2.2 Constraints

1. There will be 2 file types; System Catalog File and Data File.
2. System Catalog File is a single file named as "SystemCatalog.db"
3. **Fixed-length** records shall be used in the system.
4. The size of the **system catalog** file when it is full shall be **110 kilobytes**. It shall contain **100 pages**.
5. The size of a **system catalog page** shall be **1105 bytes**. It shall contain **5 records**. Therefore, with this design, database can store maximum **500 different types**.
6. The size of a **record** in the system catalog file shall be **221 bytes**.
7. **Data file** size shall be **130 kilobytes**. It shall contain **100 pages** and it shall only contains records of **one type**. The name of the file shall be in this format: **"TypeName.db"**.
8. With the above mentioned sizes, one data file can store maximum **1600 instances** of a record type.
9. **Data page** size shall be **1296 bytes**. It shall contain maximum **16 records**.

10. **Data record** size shall be **81 bytes**. It shall contain **20 fields** which are 4-byte integers. Not specified fields for a type shall be filled with zeros. Because we know the number of fields for a type, these zeros will not be considered at all while reading records.

11. Maximum length of a type and field name shall be **10 characters**, minimum length is **1 character**.

12. A type shall be able to have at most 20 fields, minimum 1 field.

13. Every record and page shall have a unique id.

14. When a type is deleted, all records with this type shall be also deleted.

3 Storage Structures

3.1 System Catalog

System catalog is a single file containing multiple pages. It stores metadata of the database and it has a different architecture from the data files. In this design, the system catalog file stores the name of the database and the name of the owner. Rest of the file stores the information about data types in database. The number of records that a system catalog page can store is less than the one that a data page can store, because field names cover relatively large storage size. Therefore only 5 records can be stored, otherwise the page size would be high for the buffer to handle. Following tables explains the structure.

3.2 Changes in System Catalog

System catalog file has not first page pointer anymore, because I do not need it to read the file. However, operating system has probably first page pointers for every files, therefore I do not need to redefine it again. Another change is the size of the file. I increased the page amount in catalog file, so that it can store more types now.

System Catalog File

System Catalog File (HEADER) ——————>	- NameDB (64 bytes) - OwnerName (64 bytes)
Page 1	
Page 2	
...	
Page 100	

Figure 1: First row is the header of the system catalog file. The other rows represent the pages in this file.

Page Structure in System Catalog

System Catalog Page (HEADER) ——————>					
Record 1 (HEADER) —>	- IsEmpty (1 byte) - TypeName (16 bytes) - NumFields (4 bytes)	Field 1 Name (10 bytes)	Field 2 Name (10 bytes)	...	Field 20 Name (10 bytes)
Record 2 (HEADER) —>	- IsEmpty (1 byte) - TypeName (10 bytes) - NumFields (4 bytes)	Field 1 Name (10 bytes)	Field 2 Name (10 bytes)	...	Field 20 Name (10 bytes)
...	...	...	...	...	...
Record 5 (HEADER) —>	- IsEmpty (1 byte) - TypeName (10 bytes) - NumFields (4 bytes)	Field 1 Name (10 bytes)	Field 2 Name (10 bytes)	...	Field 20 Value (10 bytes)

Figure 2: First row is the header of a system catalog page. The other rows are the records.

3.3 Data File

Data files is the main storage elements for the instances of a type in database. In my design, every type is stored in one page which is named according to the name of the type. For example, for type "Student" there is a file called "Student.db". One data file has a size of 130 kilobytes, therefore it can only store 1600 record in it. The structure of a data file is as following.

3.4 Changes in Data File

Again I increased the page number in a data file. Because I have implemented my conceptual design in project 1, it should be able to store more data and be more practical. Another change is that I removed the record id from record headers. Now, the first fields in records are used as primary keys.

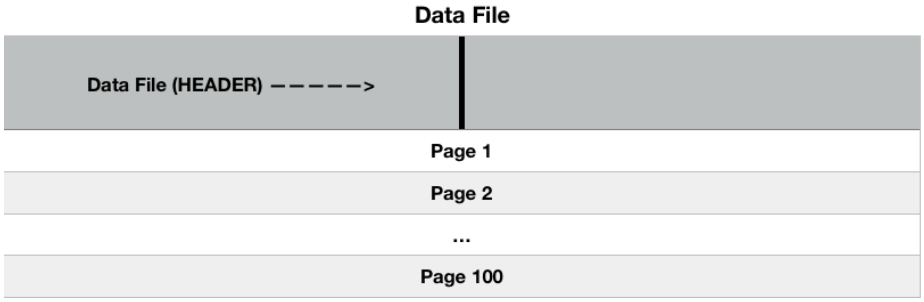

Figure 3: First row is the header of a data file. The other rows represent the pages in this file.

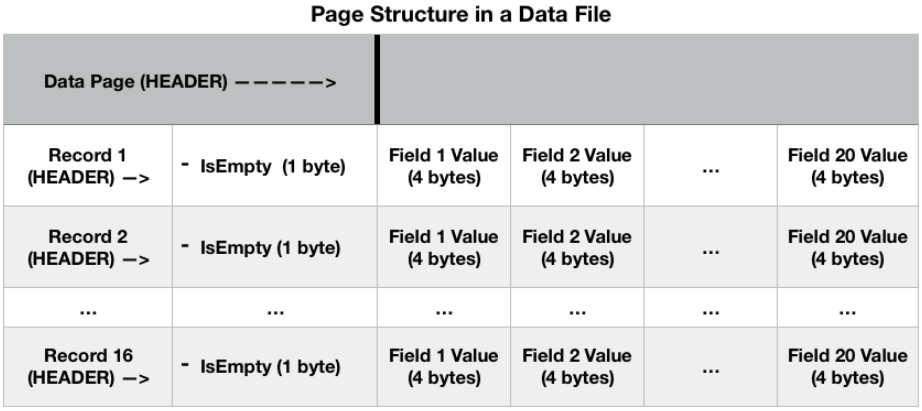

Figure 4: First row is the header of a data page. The other rows are the records.

4 Operations

4.1 DDL Operations

4.1.1 Creating a new type

When creating a new type, the system catalog file is updated by adding the information of the new type. Pages are iterated in the system catalog file with page-to-page pointers until a page with an enough space is found. Also a new data file is created with the name of the new type.

```
FUNCTION CreateType
newTypeName <- input()
newTypeFields <- input()
syscat <- open("syscat.db")
createFile("newTypeName.db")
for each page in syscat:  // Iteration is done by pointers
    if page.NumRecords < 5:
        for each record in page:
            if record.IsEmpty:
                record.TypeName <- newTypeName
                record.Fields <- newTypeFields
                record.IsEmpty <- 0
                page.NumRecords++
                syscat.NumRecords++
                return
```

4.1.2 Deleting a type

Related record in the system catalog file is found by iteration, and that record is deleted. Also the related data file is deleted.

```
FUNCTION DeleteType
typeName <- input()
syscat <- open("syscat.db")
deleteFile("typeName.db")
for each page in syscat:
    for each record in page:
        if record.TypeName == typeName:
            record.IsEmpty <- 1
            page.NumRecords--
            syscat.NumRecords--
            return
```

4.1.3 Listing all types

Only the system catalog file is examined and all types is found by iteration.

```
FUNCTION ListAllTypes
syscat <- open("syscat.db")
if syscat.NumRecords > 0:
    for each page in syscat:
        if page.NumRecords > 0:
            for each record in page:
                if record.IsEmpty == 0:
                    print(record.TypeName)
```

4.2 DML Operations

4.2.1 Creating a new record

In order to create a new record, firstly it is checked whether there exists a data file for that record type. When all of the records in a data file are deleted, data file is also deleted. Therefore it is possible that there does not exist a data file for a particular type, if there is no instance of that type. After check, available place is found and new record is written in that space.

```
FUNCTION CreateRecord
newRecordType <- input()
newRecordFields <- input()
if file("newRecordType.db") exists:
    dataFile <- open("newRecordType.db")
else:
    // Open the file, if not exists
    dataFile <- createFile("newRecordType.db")
for each page in dataFile:
    if page.NumRecords < 16:
        for each record in page:
            if record.IsEmpty:
                record.FieldValues <- newRecordFields
                record.IsEmpty <- 1
                page.NumRecords++
                dataFile.NumRecords++
                return
```

4.2.2 Deleting a new record

With the given primary key and type name, the record to be deleted is found in “recordType.db” file. The NumRecords attribute in the header of the corresponding page and file is decreased by one. Lastly, file is deleted, if there are no more records in that file.

```
FUNCTION DeleteRecord
recordPK <- input()
recordType <- input()
dataFile <- open("recordType.db")
for each page in dataFile:
    if page.NumRecords > 0:
        for each record in page:
            if record.recordID == recordPK:
                record.IsEmpty <- 1
                page.NumRecords--
                dataFile.NumRecords--
                // Delete the empty file.
                if dataFile.NumRecords == 0:
                    deleteFile(dataFile)
```

4.2.3 Searching a record

With the given primary key and type name, the record is found in the corresponding data file.

```
FUNCTION SearchRecord
recordPK <- input()
recordType <- input()
dataFile <- open("recordType.db")
for page in dataFile:
    if page.NumRecords > 0:
        for record in page:
            if record.IsEmpty == 0 and record.recordID == recordPK:
                return record
```

4.2.4 Updating a record

With the given primary key and type name, record is found with the same method as in searching. After it is found, new fields is overwritten.

```
FUNCTION UpdateRecord
recordPK <- input()
recordType <- input()
updatedRecordFields <- input()
dataFile <- open("recordType.db")
for page in dataFile:
    if page.NumRecords > 0:
        for record in page:
            if record.IsEmpty == 0 and record.recordID == recordPK:
                record.FieldValues <- updatedRecordFields
                return
```

4.2.5 Listing all records

With the given type name, the corresponding data file is opened and all records in it are printed.

```
FUNCTION ListAllRecords
recordType <- input()
dataFile <- open("recordType.db")
for page in dataFile:
    if page.NumRecords > 0:
        for record in page:
            if record.IsEmpty == 0:
                print(record)
```

5 Conclusion & Assessment

This design is a conceptual design rather than a practical design. I did not use very efficient techniques in this database design, because it is a small-scale database and it stores relatively simple objects. In contrast to inefficient implementation, my design is very simple and can be readily updated with more complex techniques.