CMPE321

Introduction to Database Systems

Storage Manager Design

Fatih İver

2016400264

Spring 2019

INTRODUCTION

This project aims to teach the internal dynamics of storage managers. Functionalities which are creating, deleting, and listing record types; creating, searching, deleting and listing records are implemented as a pseudocode to understand the how fundamental operations work in a storage manager. Since most of the design choices are done by the preparer of the project, designing such a system requires the prepare to think carefully and pay attention to details.

This implementation keeps all record types in a System Catalog file and uses a file for each record type to keep its records. Each page is 4096 Bytes which is 4 KB which is an ideal page size in practical world.

It is assumed that users will always give a valid input and disk manager exists.

ASSUMPTIONS & CONSTRAINTS

System Catalog File

- Page size is 4096 Bytes
- A page consists a page header and 15 record types
- Page header is 16 Bytes
- Page header consists page id and number of record types
- Page id is 8 Bytes
- Number of records is 8 Bytes
- Each page consists 15 record types
- Each record type is 272 Bytes in total
- Each record consists a record type and 16 record fields
- Each record type is 16 Bytes
- Each record consists of 16 record fields
- Each record field is 16 Bytes

*Each character is 1 Byte – ASCII Character

Data Files

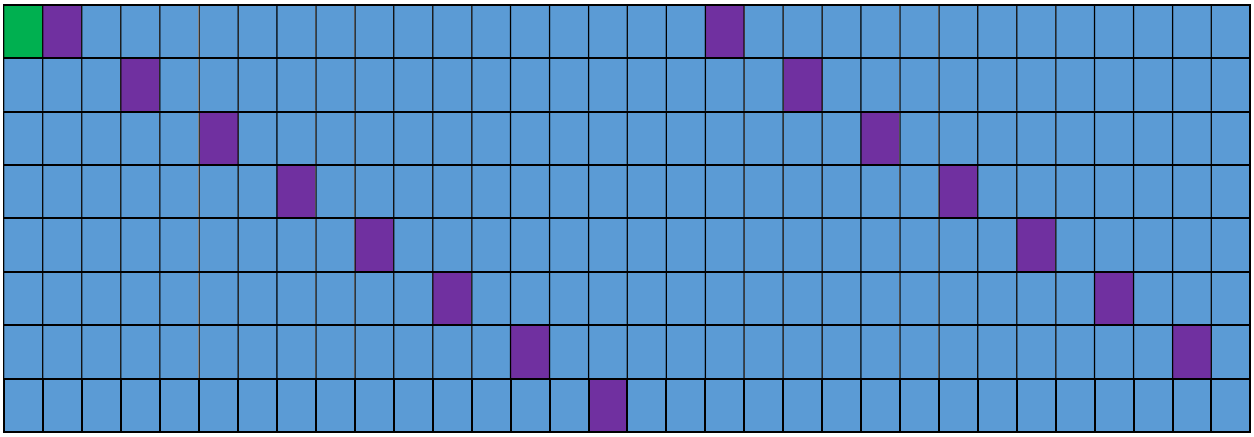
- Page size is 4096 Bytes
- Page header consists page id, a pointer to the next page, and number of records in the page
- Page header is 64 Bytes
- Page id is 16 Bytes
- Pointer to the next page is 16 Bytes
- Number of records is 32 Bytes
- Each page consists 28 record
- Records hold 4032 Bytes space in a page
- Each record consists a record id, an isEmpty field and 16 record fields
- Each record id is 8 Bytes
- Each record field is 8 Bytes
- Each isEmpty field is 8 Bytes

*Each character is 1 Byte – ASCII Character

STORAGE STRUCTURES

System Catalog File

Sample Page – Each Cell Represents 16 Bytes



Green cell represents page header which is 16 Bytes and consists page id which is 8 Bytes and number of record types which is again 8 Bytes.

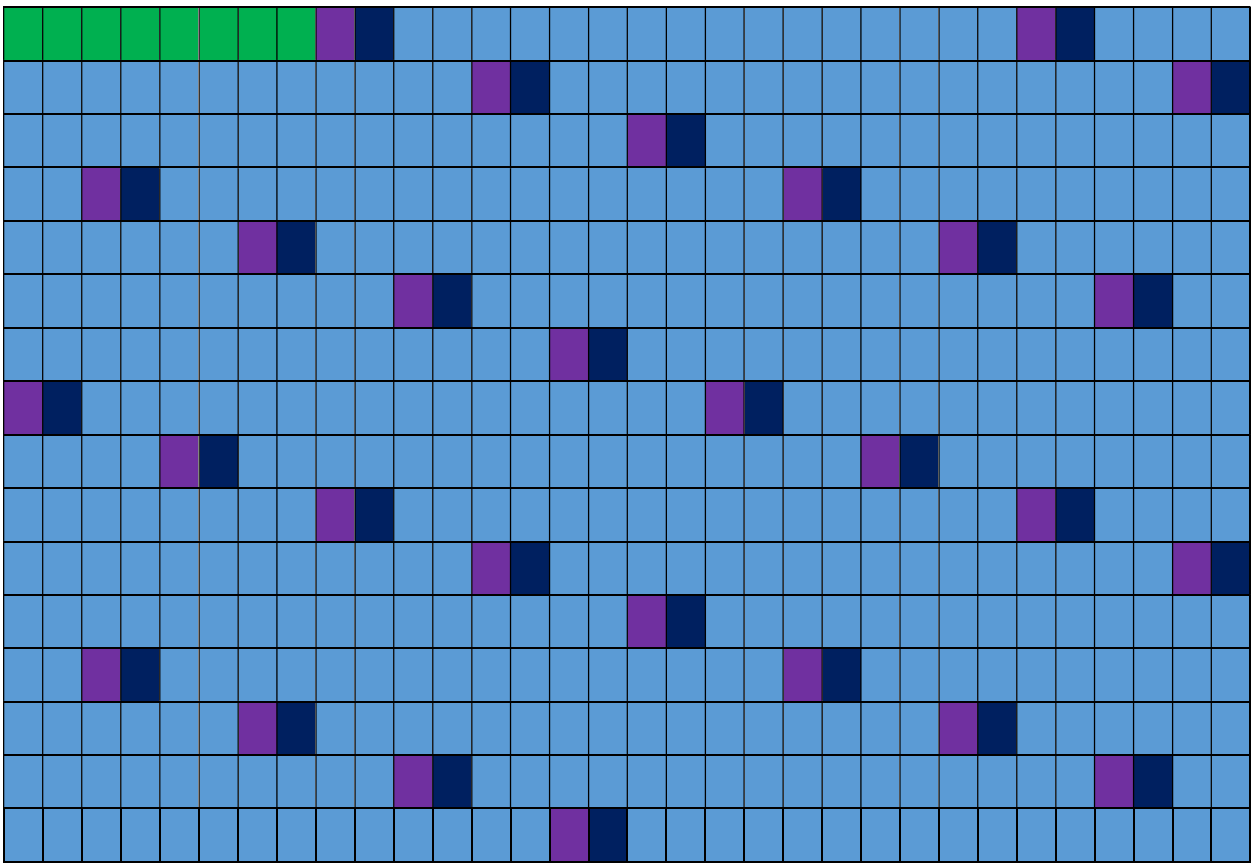
Each purple cell represents record header which is 16 Bytes and consists record type name.

Each blue cell represents record field name which is 16 Bytes.

There are 15 record types each contains 1 type and 16 field name/s.

Data Files

Sample Page – Each Cell Represents 8 Bytes



Green cell represents page header which is 64 Bytes and consists page id which is 16 Bytes and a pointer to the next page which is again 16 Bytes, and number of records which is 32 Bytes.

Each purple cell represents record header which is 8 Bytes and consists record id.

Each dark blue cell represents isEmpty field which is 8 Bytes.

Each blue cell represents record fields which are each 8 Bytes.

There are at most 28 records in total in a page.

OPERATIONS

Data Definition Language Operations

Create a Data Type

```
recordType = input()
recordFields = []
fieldNumber = int(input())
for i in range(fieldNumber):
    recordField = input()
    recordFields.append(recordField)
record = Record(recordType, recordFields)
sysCat = open("SysCat")
representation = repr(record)
sysCat.write(representation)
createDataFile(recordType)
```

Delete a Data Type

```
recordType = input()
deleteFile(recordType)
sysCat = open("sysCat")
foreach page in sysCat:
    foreach record in page:
        if record.type == recordType:
            record.type = 0
            break
```

List All Types

```
sysCat = open("sysCat")
foreach page in sysCat:
    foreach record in page:
        if record.type != 0:
            print(record.type)
```

Data Manipulation Language Operations

Create a Record

```
recordType = input()
recordFile = open(recordType)
foreach page in recordFile:
    if page.header.numberOfRecords != 28:
        page.numberOfRecords += 1
        unfullPage = page
        break
foreach record in page:
    if record.isEmpty:
        record.isEmpty = False
        fieldNumber = int(input())
        for i in range(fieldNumber):
            record.fields[i] = int(input())
        break
```

Delete a Record

```
recordType = input()
recordFile = open(recordType)
recordID = int(input())
deleted = False
foreach page in recordFile:
    foreach record in page:
        if record.ID == recordID:
            record.ID = 0
            page.numberOfRecords -= 1
            deleted = True
        break
    if deleted:
        break
```

Search for a Record

```
recordType = input()
recordFile = open(recordType)
recordID = int(input())
foreach page in recordFile:
    foreach record in page:
        if record.ID == recordID:
            return record
return None
```

List All Records of a Type

```
recordType = input()
recordFile = open(recordType)
foreach page in recordFile:
    foreach record in page:
        if not record.isEmpty:
            print(record)
```

CONCLUSIONS & ASSESMENT

Implementation of a storage manager is showed in this project. Tough it is not a practical one, it is simple to understand the concepts in database management systems. It is not practical because this storage manager keeps 16 record fields for each record types. It does not matter whether a record type needs that much field. It is assumed that all record type will have 16 record fields even they don't need it. If it is not needed, they will be stored as Null. This approach is not efficient in terms of storage, but it makes the database manager simple enough to understand and implement. Another problem with this implementation is the limit on the number of characters for the field names. Each field name is maximum 16 Bytes, since ASCII characters are used, it allows maximum 16 characters naming which is ideal because average world length in English is 8 characters. However, it limits users in terms of naming with this character limitation. If users want support for the other languages in the world, it will require to change the whole structure of the system catalog file. Another problem is search times. No order is used for insertion, since first empty slot is used. This makes searching time linear. I think such systems should provide at least logarithmic time since this is a database management system and it is possible to have so many queries in short intervals. A sorting scheme may have been used but it would make the system more complex. This implementation is made for the sake of simplicity.