CMPE321 - Summer/2020
Design of a Storage Management System

Sertay Akpınar

Contents

1. Introduction

In this project I have designed a storage management system. This system is able to perform some DDL(create, delete, list types) and DML(create, delete, search, list records) operations. I made some assumptions and design choices in order to make the system efficient. Additionally, I clarified the storage structures and the design choices with diagrams. Last but not least, I wrote the pseudocodes for DDL and DML operations.

2. Assumptions & Constraints

2.1. Assumptions

- User always enters valid input.
- All fields are integers.
- Type and field names are alphanumeric.
- A disk manager already exists that is able to fetch the necessary pages when addressed..
- Char size is 1 Byte.
- Integer size is 4 Byte.
- Page size is 1528 Byte.
- Each file contains 512 pages.
- File size is 764 KB.

2.2. Constraints

- User always enters valid input.
- A type can have at most 10 fields.
- All type names must be unique.
- The primary key of a record should be assumed to be the value of the first field of that record.
- Primary keys of each record in a given type are unique.
- Maximum length of a type and a field name is 16 characters.

3. Storage Structures

3.1. System Catalogue Design

3.1.1. System Catalogue

This is the main file of the storage system designed to store metadata. It is a mini database organized with types. It contains the type names, number of fields, and number of pages of each type. It is stored as a file named "SysCat.txt".

System Catalogue Header		
Type Names	# of Fields	# of Pages
Type 1 … Type n	int a … int b	Type 1 # of Pages … Type n # of Pages

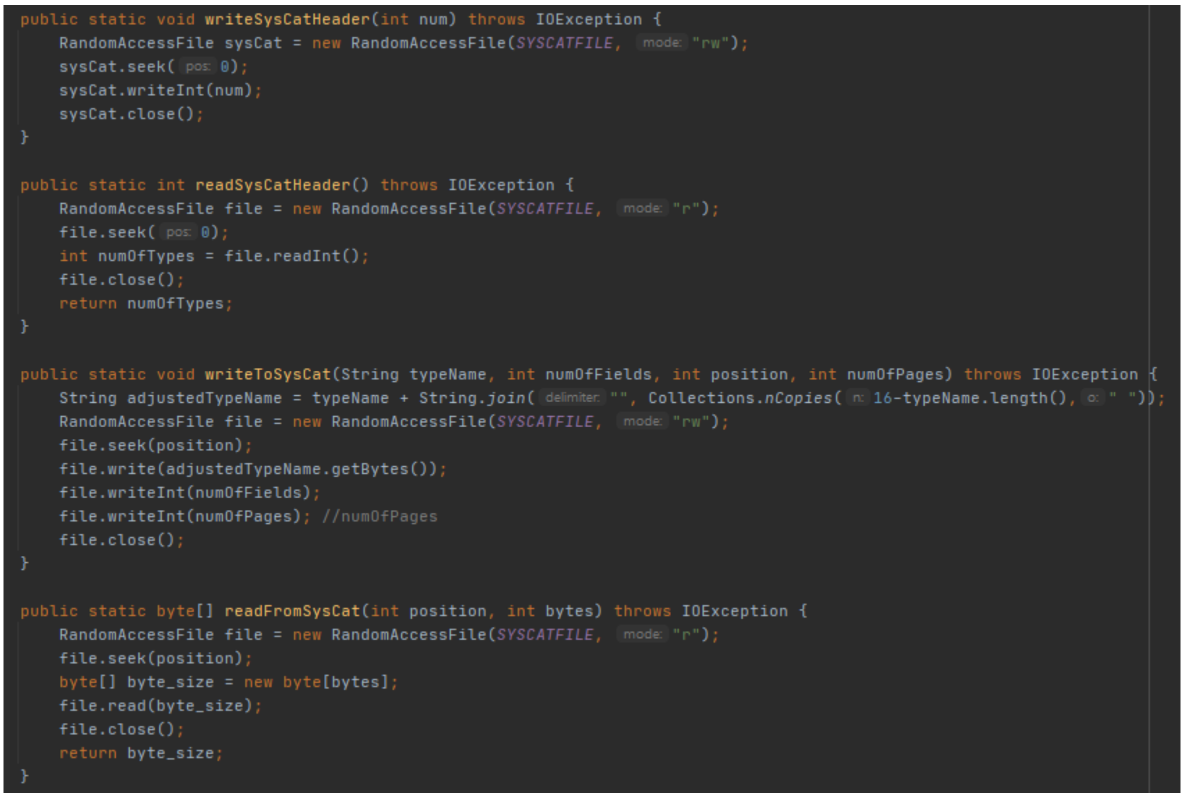
Type Names → max 16 bytes
# of Fields, # of Pages → 4 bytes

3.1.2. System Catalogue Header

This is the header of the system catalogue which contains the number of types.

Number of Types

Number of Types → 4 bytes



```
public static void writeSysCatHeader(int num) throws IOException {
    RandomAccessFile sysCat = new RandomAccessFile(SYSCATFILE,  mode: "rw");
    sysCat.seek( pos: 0);
    sysCat.writeInt(num);
    sysCat.close();
}

public static int readSysCatHeader() throws IOException {
    RandomAccessFile file = new RandomAccessFile(SYSCATFILE,  mode: "r");
    file.seek( pos: 0);
    int numOfTypes = file.readInt();
    file.close();
    return numOfTypes;
}

public static void writeToSysCat(String typeName, int numOfFields, int position, int numOfPages) throws IOException {
    String adjustedTypeName = typeName + String.join( delimiter: "", Collections.nCopies( n: 16-typeName.length(),  o: " "));
    RandomAccessFile file = new RandomAccessFile(SYSCATFILE,  mode: "rw");
    file.seek(position);
    file.write(adjustedTypeName.getBytes());
    file.writeInt(numOfFields);
    file.writeInt(numOfPages); //numOfPages
    file.close();
}

public static byte[] readFromSysCat(int position, int bytes) throws IOException {
    RandomAccessFile file = new RandomAccessFile(SYSCATFILE,  mode: "r");
    file.seek(position);
    byte[] byte_size = new byte[bytes];
    file.read(byte_size);
    file.close();
    return byte_size;
}
```

3.2. File Design

Each file is specific to a type and the file contains all of the pages of that specific type. Each type's file can be accessed from the system catalogue with the type name. Each file consists of at most 512 pages.

Page 1 ... Page n

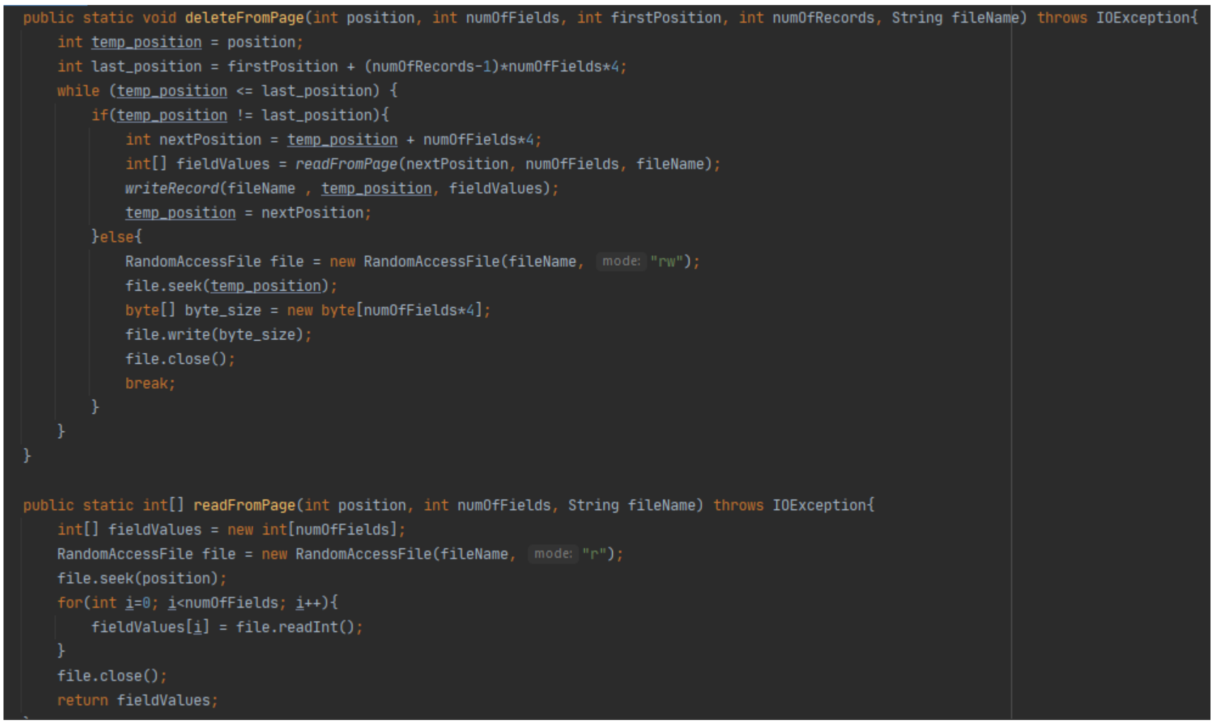
3.3. Page Design

3.3.1. Page

Pages are stored in files and contain the records of a type. A page(1540 byte) can have at least 38 records(each record has 10 fields) and at max 380 records(each record has 1 one field).

Page Header

Record 1 … … Record n



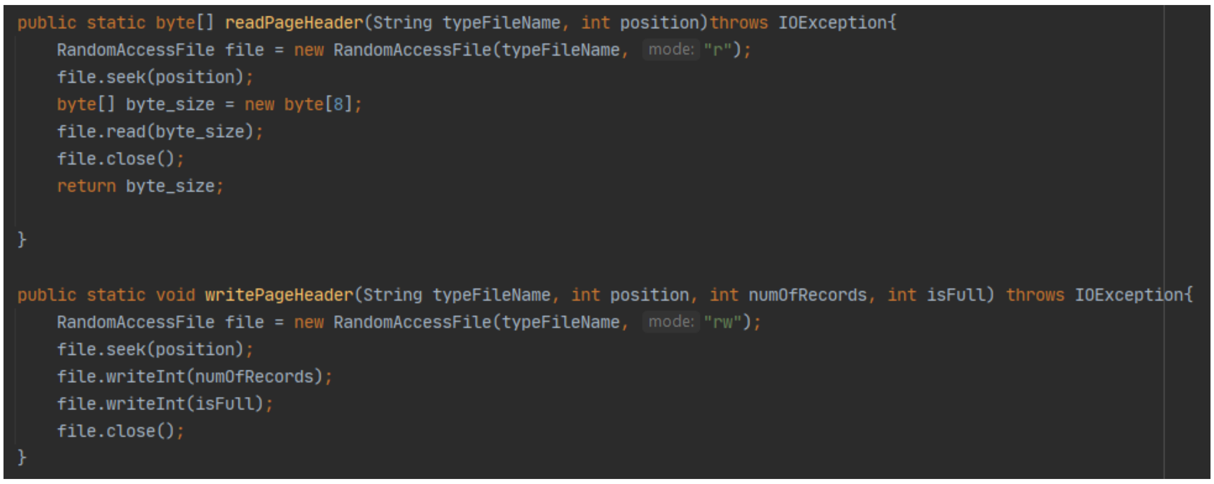
```java
public static void deleteFromPage(int position, int numOfFields, int firstPosition, int numOfRecords, String fileName) throws IOException{
    int temp_position = position;
    int last_position = firstPosition + (numOfRecords-1)*numOfFields*4;
    while (temp_position <= last_position) {
        if(temp_position != last_position){
            int nextPosition = temp_position + numOfFields*4;
            int[] fieldValues = readFromPage(nextPosition, numOfFields, fileName);
            writeRecord(fileName , temp_position, fieldValues);
            temp_position = nextPosition;
        }else{
            RandomAccessFile file = new RandomAccessFile(fileName,  mode: "rw");
            file.seek(temp_position);
            byte[] byte_size = new byte[numOfFields*4];
            file.write(byte_size);
            file.close();
            break;
        }
    }
}

public static int[] readFromPage(int position, int numOfFields, String fileName) throws IOException{
    int[] fieldValues = new int[numOfFields];
    RandomAccessFile file = new RandomAccessFile(fileName,  mode: "r");
    file.seek(position);
    for(int i=0; i<numOfFields; i++){
        fieldValues[i] = file.readInt();
    }
    file.close();
    return fieldValues;
```

3.3.2. Page Header

Each page keeps the next page ID. Pages are linked to each other to easily bring the next page to RAM. Pages are the structures that are loaded to RAM one by one. Every page has the id of the next page to easily bring the next page to RAM.

# of Records	isFull

Both of them are 4 bytes. Total size is 8 bytes.(isFull is a binary value.)



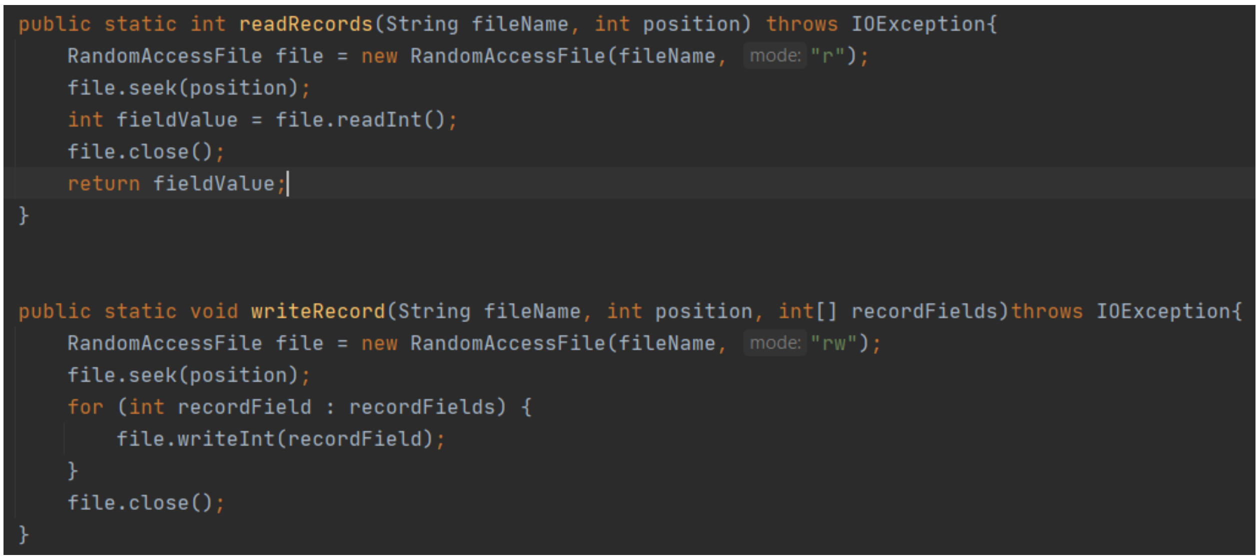
```java
public static byte[] readPageHeader(String typeFileName, int position)throws IOException{
    RandomAccessFile file = new RandomAccessFile(typeFileName,  mode: "r");
    file.seek(position);
    byte[] byte_size = new byte[8];
    file.read(byte_size);
    file.close();
    return byte_size;

}

public static void writePageHeader(String typeFileName, int position, int numOfRecords, int isFull) throws IOException{
    RandomAccessFile file = new RandomAccessFile(typeFileName,  mode: "rw");
    file.seek(position);
    file.writeInt(numOfRecords);
    file.writeInt(isFull);
    file.close();
}
```

3.4. Record Design

Record does not have a header. It can have at most 10 field. Field 0 is the primary key for all instances and all fields are integers. Every field in a type is unique.

Field 0 (Primary Key)	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9

Field → 4 bytes

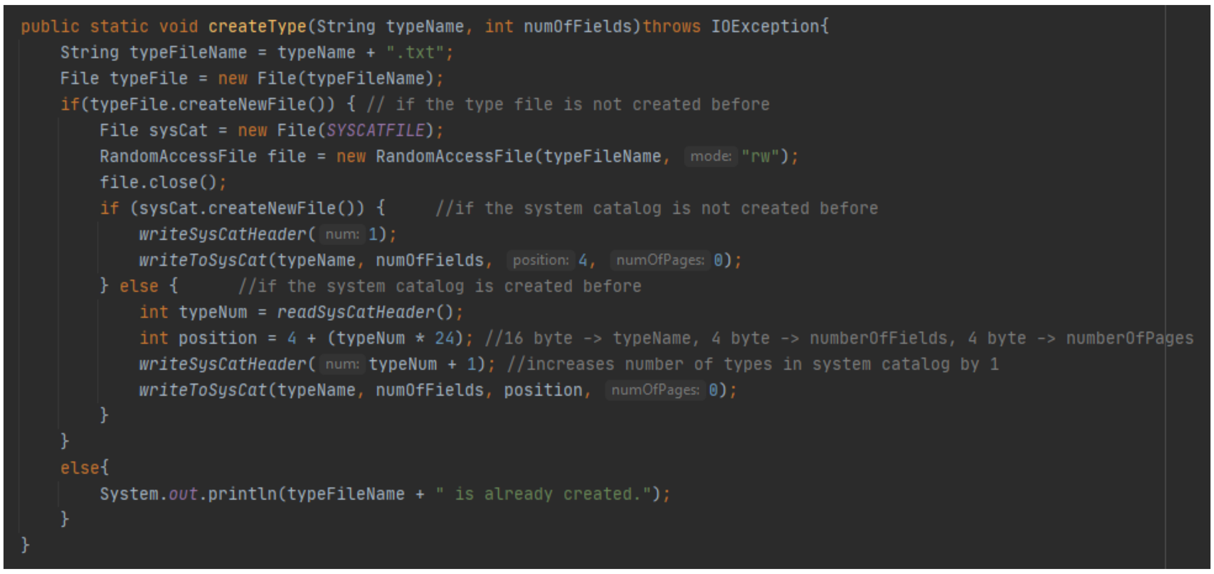


```java
public static int readRecords(String fileName, int position) throws IOException{
    RandomAccessFile file = new RandomAccessFile(fileName,  mode: "r");
    file.seek(position);
    int fieldValue = file.readInt();
    file.close();
    return fieldValue;
}


public static void writeRecord(String fileName, int position, int[] recordFields)throws IOException{
    RandomAccessFile file = new RandomAccessFile(fileName,  mode: "rw");
    file.seek(position);
    for (int recordField : recordFields) {
        file.writeInt(recordField);
    }
    file.close();
}
```

4. Operations

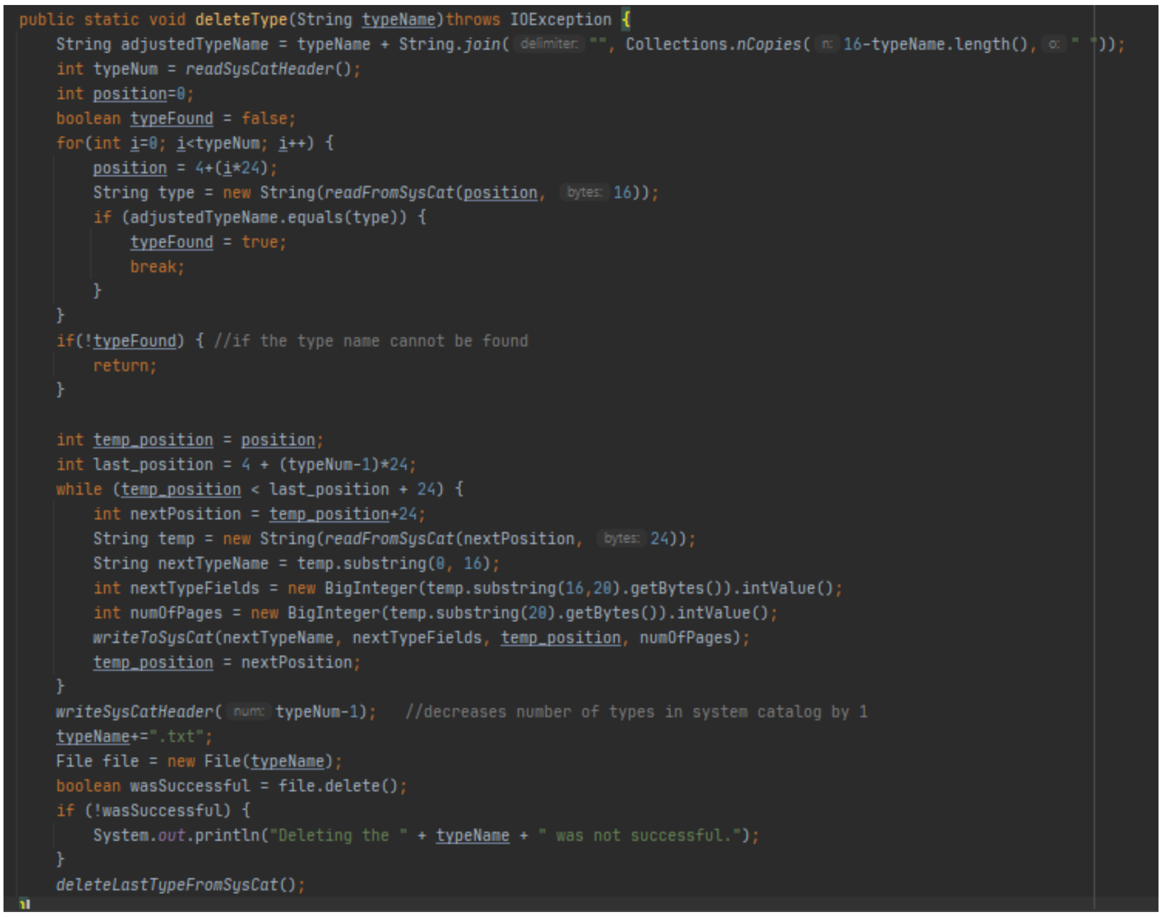
4.1. DDL Operations

4.1.1. Create a Type



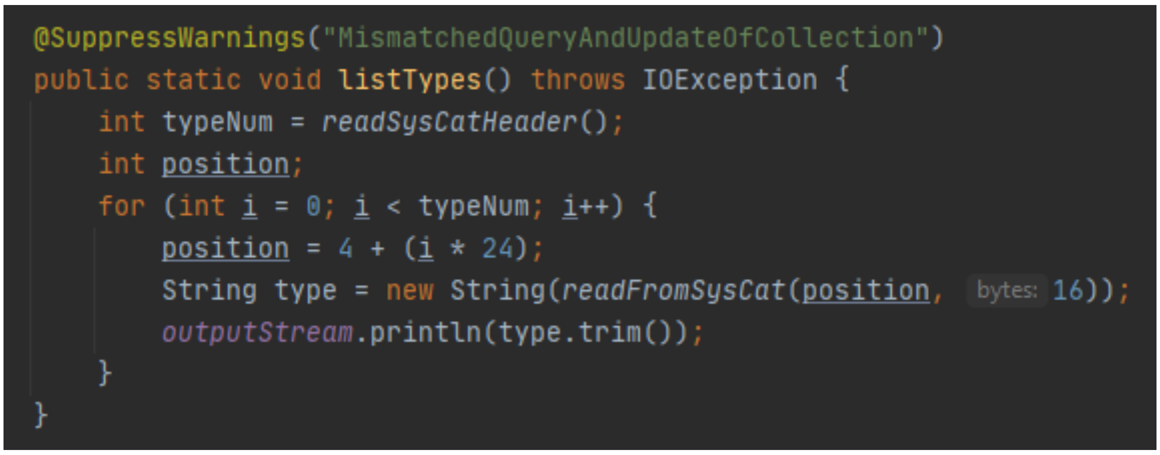
```java
public static void createType(String typeName, int numOfFields)throws IOException{
    String typeFileName = typeName + ".txt";
    File typeFile = new File(typeFileName);
    if(typeFile.createNewFile()) { // if the type file is not created before
        File sysCat = new File(SYSCATFILE);
        RandomAccessFile file = new RandomAccessFile(typeFileName,  mode: "rw");
        file.close();
        if (sysCat.createNewFile()) {     //if the system catalog is not created before
            writeSysCatHeader( num: 1);
            writeToSysCat(typeName, numOfFields,  position: 4,  numOfPages: 0);
        } else {      //if the system catalog is created before
            int typeNum = readSysCatHeader();
            int position = 4 + (typeNum * 24); //16 byte -> typeName, 4 byte -> numberOfFields, 4 byte -> numberOfPages
            writeSysCatHeader( num: typeNum + 1); //increases number of types in system catalog by 1
            writeToSysCat(typeName, numOfFields, position,  numOfPages: 0);
        }
    }
    else{
        System.out.println(typeFileName + " is already created.");
    }
}
```

4.1.2. Delete a Type



```java
public static void deleteType(String typeName)throws IOException {
    String adjustedTypeName = typeName + String.join( delimiter: "", Collections.nCopies( n: 16-typeName.length(),  o: " "));
    int typeNum = readSysCatHeader();
    int position=0;
    boolean typeFound = false;
    for(int i=0; i<typeNum; i++) {
        position = 4+(i*24);
        String type = new String(readFromSysCat(position,  bytes: 16));
        if (adjustedTypeName.equals(type)) {
            typeFound = true;
            break;
        }
    }
    if(!typeFound) { //if the type name cannot be found
        return;
    }

    int temp_position = position;
    int last_position = 4 + (typeNum-1)*24;
    while (temp_position < last_position + 24) {
        int nextPosition = temp_position+24;
        String temp = new String(readFromSysCat(nextPosition,  bytes: 24));
        String nextTypeName = temp.substring(0, 16);
        int nextTypeFields = new BigInteger(temp.substring(16,20).getBytes()).intValue();
        int numOfPages = new BigInteger(temp.substring(20).getBytes()).intValue();
        writeToSysCat(nextTypeName, nextTypeFields, temp_position, numOfPages);
        temp_position = nextPosition;
    }
    writeSysCatHeader( num: typeNum-1);   //decreases number of types in system catalog by 1
    typeName+=".txt";
    File file = new File(typeName);
    boolean wasSuccessful = file.delete();
    if (!wasSuccessful) {
        System.out.println("Deleting the " + typeName + " was not successful.");
    }
    deleteLastTypeFromSysCat();
}
```

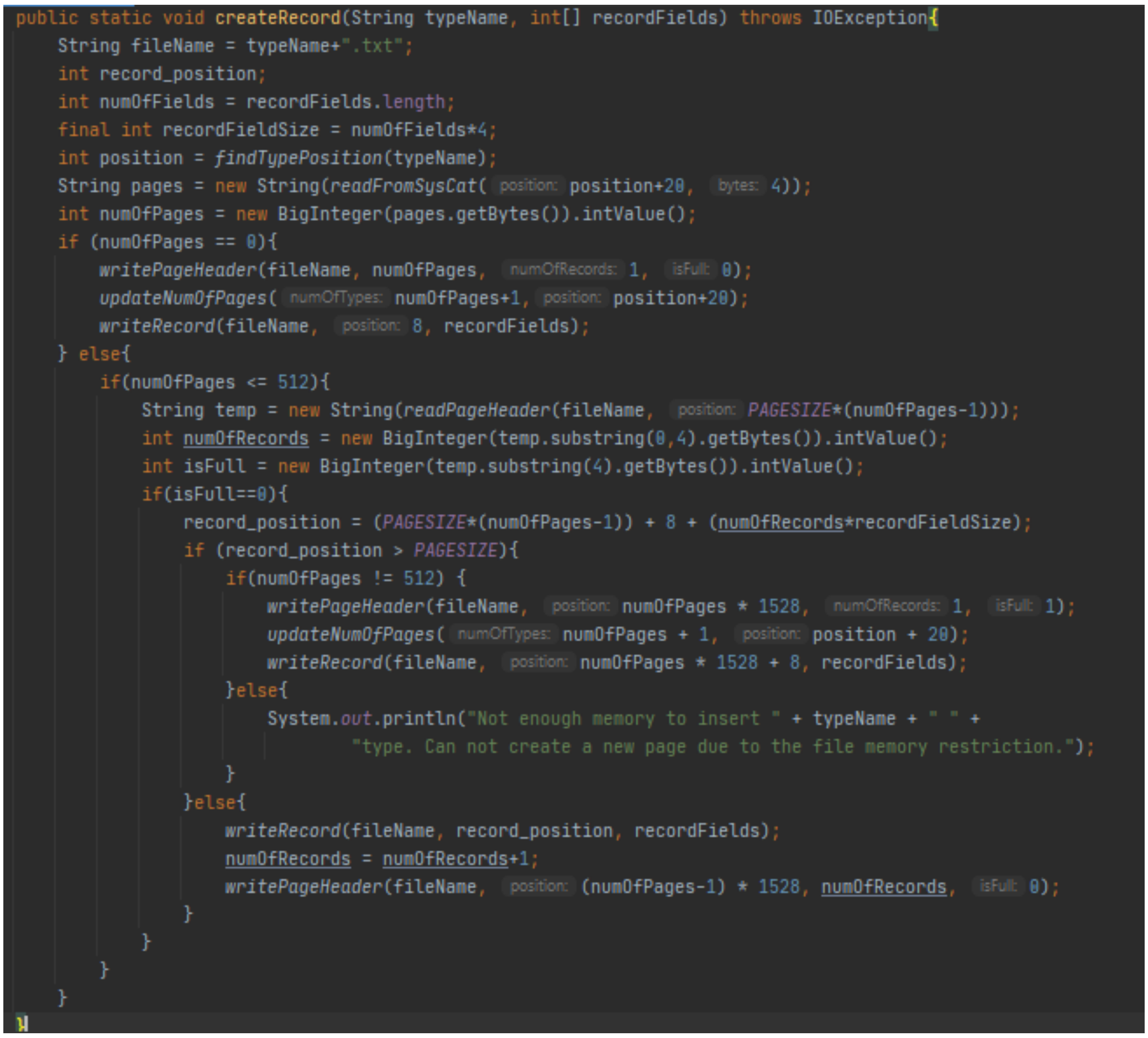
4.1.3. List All Types



```
@SuppressWarnings("MismatchedQueryAndUpdateOfCollection")
public static void listTypes() throws IOException {
    int typeNum = readSysCatHeader();
    int position;
    for (int i = 0; i < typeNum; i++) {
        position = 4 + (i * 24);
        String type = new String(readFromSysCat(position, bytes: 16));
        outputStream.println(type.trim());
    }
}
```

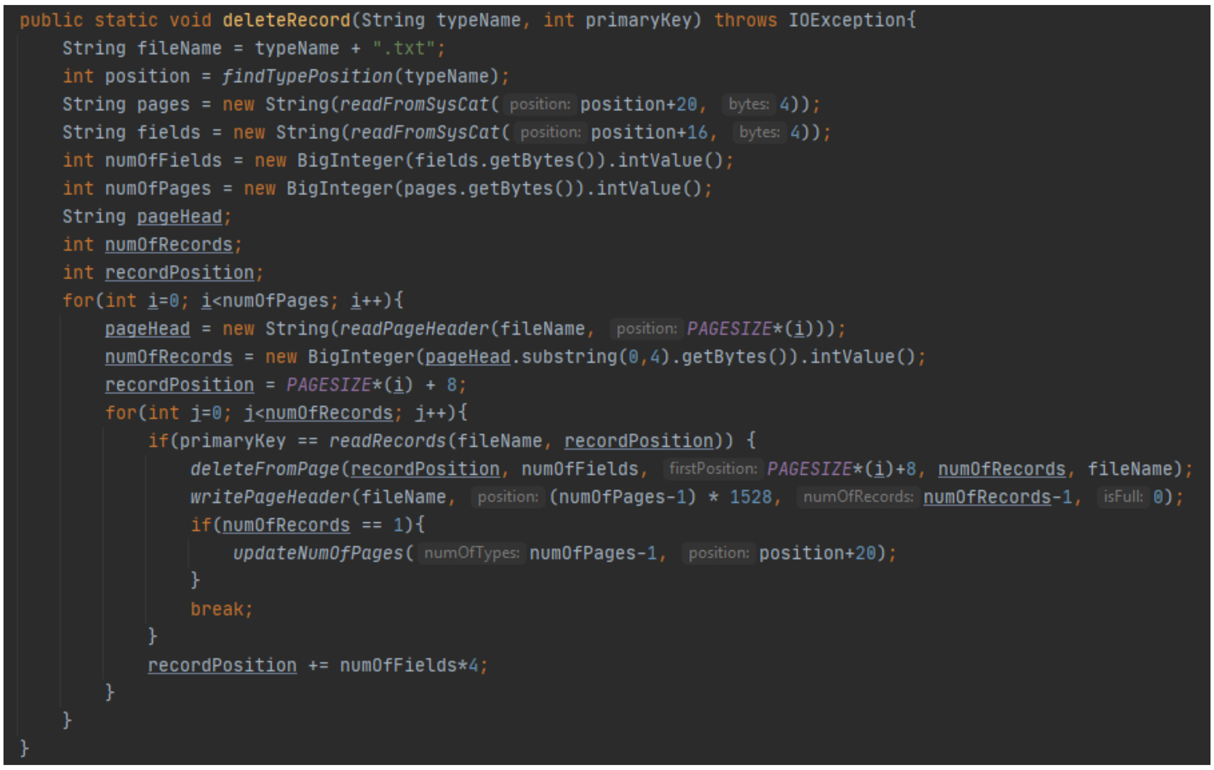
4.2. DML Operations

4.2.1. Create a Record



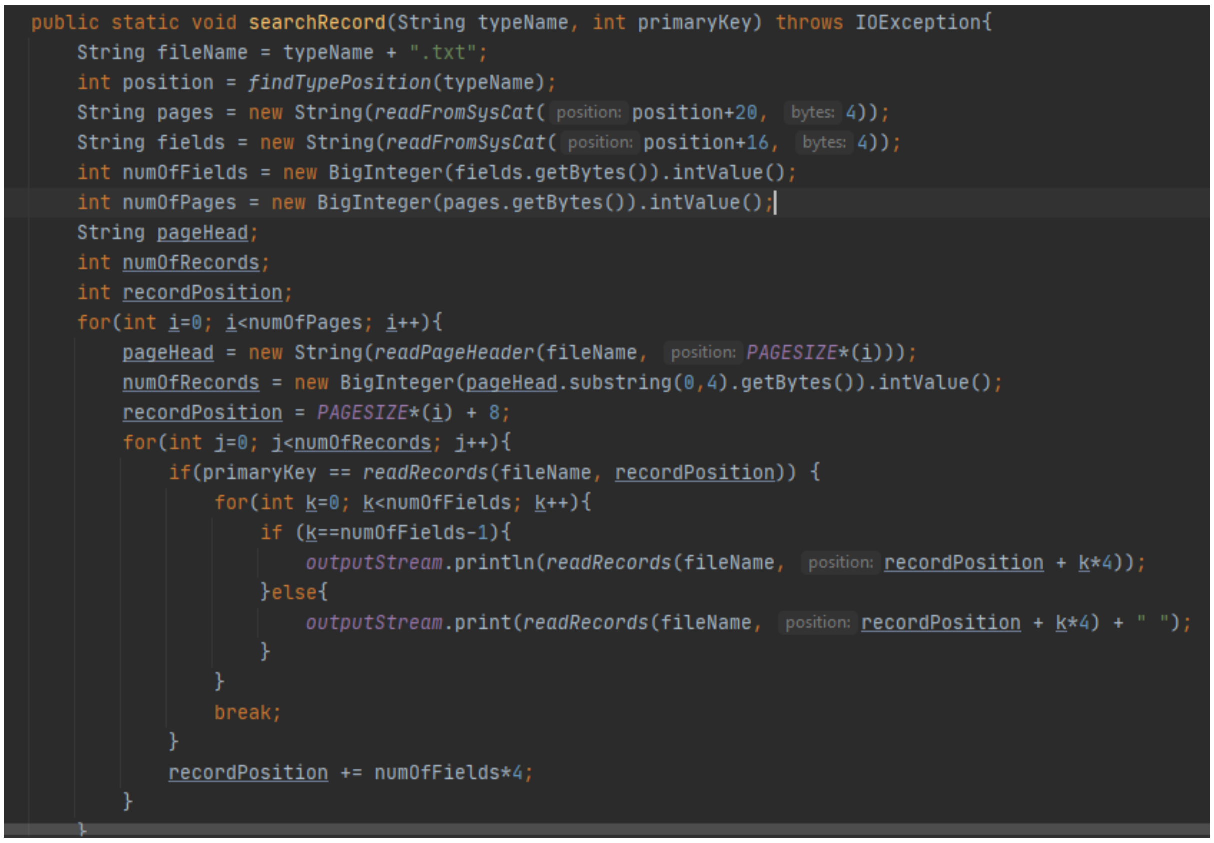
```
public static void createRecord(String typeName, int[] recordFields) throws IOException{
    String fileName = typeName+".txt";
    int record_position;
    int numOfFields = recordFields.length;
    final int recordFieldSize = numOfFields*4;
    int position = findTypePosition(typeName);
    String pages = new String(readFromSysCat( position: position+20,  bytes: 4));
    int numOfPages = new BigInteger(pages.getBytes()).intValue();
    if (numOfPages == 0){
        writePageHeader(fileName, numOfPages,  numOfRecords: 1,  isFull: 0);
        updateNumOfPages( numOfTypes: numOfPages+1,  position: position+20);
        writeRecord(fileName,  position: 8, recordFields);
    } else{
        if(numOfPages <= 512){
            String temp = new String(readPageHeader(fileName,  position: PAGESIZE*(numOfPages-1)));
            int numOfRecords = new BigInteger(temp.substring(0,4).getBytes()).intValue();
            int isFull = new BigInteger(temp.substring(4).getBytes()).intValue();
            if(isFull==0){
                record_position = (PAGESIZE*(numOfPages-1)) + 8 + (numOfRecords*recordFieldSize);
                if (record_position > PAGESIZE){
                    if(numOfPages != 512) {
                        writePageHeader(fileName,  position: numOfPages * 1528,  numOfRecords: 1,  isFull: 1);
                        updateNumOfPages( numOfTypes: numOfPages + 1,  position: position + 20);
                        writeRecord(fileName,  position: numOfPages * 1528 + 8, recordFields);
                    }else{
                        System.out.println("Not enough memory to insert " + typeName + " " +
                                "type. Can not create a new page due to the file memory restriction.");
                    }
                }else{
                    writeRecord(fileName, record_position, recordFields);
                    numOfRecords = numOfRecords+1;
                    writePageHeader(fileName,  position: (numOfPages-1) * 1528, numOfRecords,  isFull: 0);
                }
            }
        }
    }
}
```

4.2.2. Delete a Record



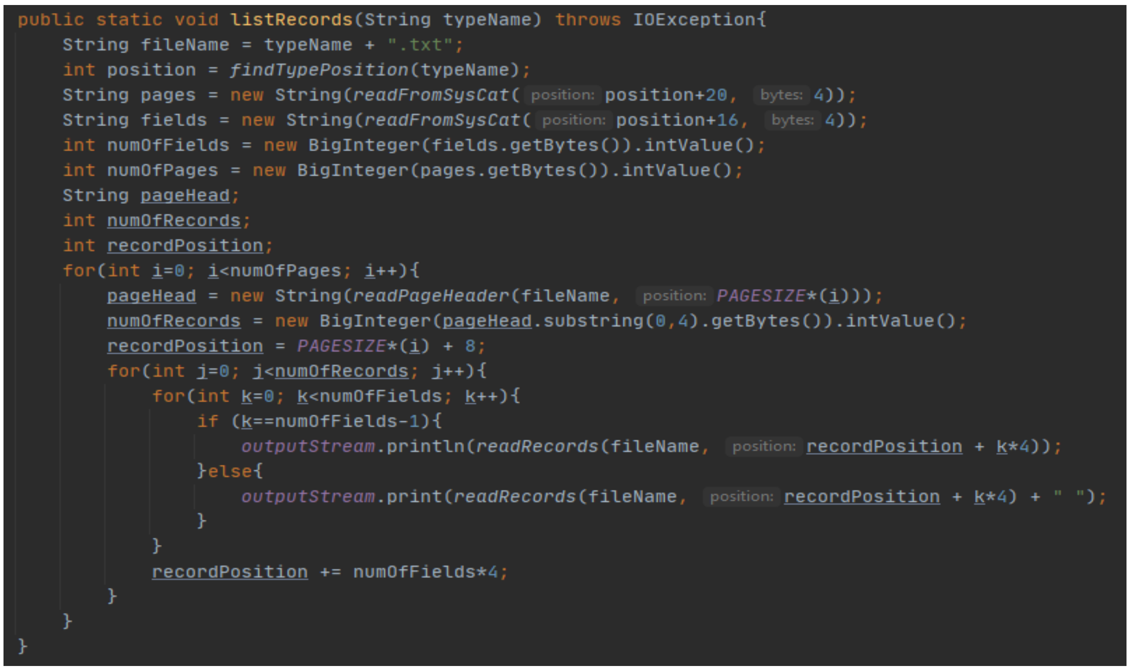
```java
public static void deleteRecord(String typeName, int primaryKey) throws IOException{
    String fileName = typeName + ".txt";
    int position = findTypePosition(typeName);
    String pages = new String(readFromSysCat( position: position+20,  bytes: 4));
    String fields = new String(readFromSysCat( position: position+16,  bytes: 4));
    int numOfFields = new BigInteger(fields.getBytes()).intValue();
    int numOfPages = new BigInteger(pages.getBytes()).intValue();
    String pageHead;
    int numOfRecords;
    int recordPosition;
    for(int i=0; i<numOfPages; i++){
        pageHead = new String(readPageHeader(fileName,  position: PAGESIZE*(i)));
        numOfRecords = new BigInteger(pageHead.substring(0,4).getBytes()).intValue();
        recordPosition = PAGESIZE*(i) + 8;
        for(int j=0; j<numOfRecords; j++){
            if(primaryKey == readRecords(fileName, recordPosition)) {
                deleteFromPage(recordPosition, numOfFields,  firstPosition: PAGESIZE*(i)+8, numOfRecords, fileName);
                writePageHeader(fileName,  position: (numOfPages-1) * 1528,  numOfRecords: numOfRecords-1,  isFull: 0);
                if(numOfRecords == 1){
                    updateNumOfPages( numOfTypes: numOfPages-1,  position: position+20);
                }
                break;
            }
            recordPosition += numOfFields*4;
        }
    }
}
```

4.2.3. Search for a Record



```java
public static void searchRecord(String typeName, int primaryKey) throws IOException{
    String fileName = typeName + ".txt";
    int position = findTypePosition(typeName);
    String pages = new String(readFromSysCat( position: position+20,  bytes: 4));
    String fields = new String(readFromSysCat( position: position+16,  bytes: 4));
    int numOfFields = new BigInteger(fields.getBytes()).intValue();
    int numOfPages = new BigInteger(pages.getBytes()).intValue();
    String pageHead;
    int numOfRecords;
    int recordPosition;
    for(int i=0; i<numOfPages; i++){
        pageHead = new String(readPageHeader(fileName,  position: PAGESIZE*(i)));
        numOfRecords = new BigInteger(pageHead.substring(0,4).getBytes()).intValue();
        recordPosition = PAGESIZE*(i) + 8;
        for(int j=0; j<numOfRecords; j++){
            if(primaryKey == readRecords(fileName, recordPosition)) {
                for(int k=0; k<numOfFields; k++){
                    if (k==numOfFields-1){
                        outputStream.println(readRecords(fileName,  position: recordPosition + k*4));
                    }else{
                        outputStream.print(readRecords(fileName,  position: recordPosition + k*4) + " ");
                    }
                }
                break;
            }
            recordPosition += numOfFields*4;
        }
    }
```

4.2.4. List All Records of a Type



```java
public static void listRecords(String typeName) throws IOException{
    String fileName = typeName + ".txt";
    int position = findTypePosition(typeName);
    String pages = new String(readFromSysCat( position: position+20,  bytes: 4));
    String fields = new String(readFromSysCat( position: position+16,  bytes: 4));
    int numOfFields = new BigInteger(fields.getBytes()).intValue();
    int numOfPages = new BigInteger(pages.getBytes()).intValue();
    String pageHead;
    int numOfRecords;
    int recordPosition;
    for(int i=0; i<numOfPages; i++){
        pageHead = new String(readPageHeader(fileName,  position: PAGESIZE*(i)));
        numOfRecords = new BigInteger(pageHead.substring(0,4).getBytes()).intValue();
        recordPosition = PAGESIZE*(i) + 8;
        for(int j=0; j<numOfRecords; j++){
            for(int k=0; k<numOfFields; k++){
                if (k==numOfFields-1){
                    outputStream.println(readRecords(fileName,  position: recordPosition + k*4));
                }else{
                    outputStream.print(readRecords(fileName,  position: recordPosition + k*4) + " ");
                }
            }
            recordPosition += numOfFields*4;
        }
    }
}
```

5. Conclusion & Assessment

In this project, I have designed a simple storage management system and documented my design according to the requirements and constraints mentioned in the project description. My main goal while doing this project was to understand the logic behind the database management system and have knowledge on the structures and the algorithms I used. At the same time, I have tried to design my system as efficiently as possible. I used dynamic memory allocation in page design. It makes my system flexible, especially in choosing the number of fields of a record. However, I believe that there some certain things need to be improved for future development. One of them is, each file is type specific. I think that it would be better if I change it in development. Since a type can have data on different files, this restriction is kind of an obstacle to make my system efficient. Lastly, I used some methods whose declarations are not given in the pseudo-codes. From my opinion, since their functions in the code are quite intuitive with their names I used them without defining them explicitly.