МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем

Отчет   
по лабораторной работе №1  
на тему: « Организация файла в виде кучи»  
по дисциплине «Базы данных»

Выполнили: Марочкин М.А. Шифр: 170584   
 Шорин В.Д. Шифр: 171406  
 Щекотихин С.Е. Шифр: 170590  
ИПАИТ  
Направление: 09.03.04 «Программная инженерия»  
Группа: 71-ПГ  
Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Отметка о зачете:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата «\_\_\_\_» \_\_\_\_\_\_ 2019 г.

Орел, 2019 г.

**Цель работы:**

Написать программу, которая работает с организованным в виде кучи файлом, хранящем информацию об отношении «студент».

В программе должны быть реализованы следующие функции:

* добавление информации о студенте;
* изменение информации о студенте;
* удаление информации о студенте;
* осуществление поиска информации о студенте.

Отношение студент должно содержать следующие атрибуты: номер зачетки (тип integer), фамилия (тип string (30)), имя (тип string (20)), отчество (тип string (30)), номер группы (тип integer).

**Контрольные вопросы**

1. **Что такое запись?**

Для N-арного отношения запись представляет собой упорядоченный набор из N значений, по одному значению для каждого поля отношения.

1. **Какие дополнительные байты может содержать запись?**

Запись включает столько байт, сколько требуется для хранения каждого поля в зависимости от его типа. Кроме того, запись может включать дополнительные байты, не относящиеся к полям. Это необходимо в следующих случаях:

1. Некоторые байты могут свидетельствовать о формате записи. Это необходимо, если совместно могут храниться записи различных форматов.

2. Один или несколько байт могут хранить информацию о длине записи. Это необходимо, если записи могут иметь переменную длину. В записях с постоянной длиной такая информация не храниться, а вычисляется исходя из типов полей.

3. Один бит может отводиться под информацию о том является ли запись удаленной или нет. В некоторых БД записи физически не удаляются, а соответствующий бит переводится в состояние «запись удалена». Такие записи продолжают храниться, но не рассматриваются при построении запросов. В этом случае СУБД должна иметь функцию сборки мусора. Этот процесс запускается в некоторый момент времени и перераспределяет записи, физически удаляя помеченные записи.

4. Бит использования. Он хранит информацию о наличии свободного места.

5. Неиспользуемое пространство. Байты, не хранящие никакой информации. Такие байты необходимы для более удобного чтения записей. Например, для достижения кратности разрядности процессора.

Если запись подразумевает такие типы данных как text, image, blob (двоичная информация), то такие данные не хранятся вместе с записью, а хранятся в отдельном блоке. В записи же хранится адрес.

1. **Что такое блок?**

Наиболее быстрая обработка данных была бы возможна, если размещать БД целиком в оперативной памяти. Однако практически размеры БД гораздо больше, поэтому возникает необходимость переноса фрагментов данных с диска (внешней памяти) в оперативную (первичную) память. Поэтому можно говорить, что данные на диске хранятся разделенными на части. Такие части называются ***блоками*** или ***страницами***. Размер блока зависит от количества байт, которые считываются с диска за один раз.

1. **В чем особенности организации файлов в виде кучи?**

Это наиболее тривиальный способ организации файлов. При этом записи записываются в блоки, без упорядочивания по какому-либо признаку. Сами блоки выстраиваются таким же образом. То есть, записи помещаются в один блок последовательно по мере появления. Когда блок заполняется, то выделяется новый блок, указатель на него помещается в предыдущий блок и аналогичным образом производится заполнение нового блока.

1. **Эффективность рабы с файлом, организованным в виде кучи.**

Пусть в файле существует ***N*** записей. В каждый блок может поместиться ***R*** записей. То есть в файле должно быть ***N/R*** блоков. Время выполнения операций измеряем количеством блоков, которые должны быть запрошены с диска или записаны на него.

При **поиске** записи в файле возможны два варианта:

1) N/R блоков придется просмотреть, если запись отсутствует в файле или, если ключевое поле, не является уникальным;

2) в противном случае потребуется считать ***N/2R*** блоков.

При **вставке** необходимо запросить последнюю запись кучи, так как именно после нее идет свободное место. То есть нужно считать последний блок. Если последний блок не имеет пустого места, то нужно начать новый блок. После вставки блок должен быть записан на диск. Таким образом, для вставки потребуется два обращения к диску – один по чтению, другой - по записи. Если необходимо контролировать отсутствие дублирующих записей, то время выполнения корректируется на время выполнения поиска.

**Удаление** требует найти запись, то есть выполнить операцию поиска, а затем выполнить операцию записи блока, в котором была найдена запись. Таким образом, для удаления записи потребуется в среднем ***((N/2R)+1)*** доступов, если запись существует в файле. Если ее нет в файле, то выполниться N/R доступов. Для дальнейшего использования освободившегося пространства последняя запись последнего блока может быть перемещена на место удаленной записи. Это немного замедлит процесс удаления, но уменьшит количество блоков, что в дальнейшем ускорит выполнение всех операций.

Время выполнения **модификации** аналогично времени выполнения операции удаления.

Таким образом, основное время при выполнении всех операций затрачивается на поиск записи. Следовательно, необходимо ускорить именно поиск. Это может быть достигнуто за счет другой организации файла.

**Код**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "bits/stdc++.h" #include <Windows.h>

using namespace std;FILE \*file ;string fileName;

struct Zap {int studentId = -1;char secondName[30]; char firstName[20];char midName[30];int groupId = -1;};

struct Block {Zap records[5];};

void CreateFile() {

if ((file = fopen(fileName.c\_str(), "rb+")) == NULL) {

if((file=fopen(fileName.c\_str(),"wb+"))==NULL){cout <<"Cant create and open the file" << endl;

system("pause"); exit(-1);}fclose(file);

if((file=fopen(fileName.c\_str(),"rb+"))==NULL) {cout << "Cant create and open the file" << endl;

system("pause"); exit(-1); }

}

}

void ShowZap(Zap &zap) {cout << "Student Id: " << zap.studentId << endl << "Second Name: " << zap.secondName << endl << "First Name: " << zap.firstName << endl<< "Mid Name: " << zap.midName << endl << "Group Id: " << zap.groupId << endl << endl;}

int SearchStudentInDB(int inStudentId, Block &block, uint64\_t &offset) {

fseek(file, 0, SEEK\_SET);offset = 0;

while(fread(&block,sizeof(Block),1,file)){for(int i=0;i<5;i++){if(block.records[i].studentId == inStudentId){

return i; }

}offset = ftell(file);

}return -1;

}

bool ShowDB() {

bool isDBExist = false; Block block; fseek(file, 0, SEEK\_SET);

while (fread(&block, sizeof(Block), 1, file)) {

for (int i = 0; i < 5; i++){if (block.records[i].studentId!=-1){ShowZap(block.records[i]);isDBExist = true;}}

}return isDBExist;

}

void AddStudentInDB() {

Block block;Zap curStudentZap;system("cls");cout << "Enter student id: ";cin >> curStudentZap.studentId;

cout << "Second name: "; cin >> curStudentZap.secondName;cout << "First name: ";

cin >> curStudentZap.firstName; cout << "Mid name: "; cin >> curStudentZap.midName;

cout << "Group id: "; cin >> curStudentZap.groupId; cout << endl << endl;

uint64\_t t\_off; if (SearchStudentInDB(curStudentZap.studentId, block, t\_off) != -1) { cout << "This student id already exist" << endl; system("pause"); return; }

fseek(file, -(long long int)sizeof(Block), SEEK\_END);fread(&block, sizeof(Block), 1, file);

if (block.records[4].studentId != -1) {block = Block();block.records[0] = curStudentZap;fseek(file, 0, SEEK\_END); }

else { int i = -1;while (block.records[++i].studentId > -1); block.records[i] = curStudentZap;

fseek(file, -(long long int)sizeof(Block), SEEK\_END); }

fwrite(&block, sizeof(Block), 1, file);cout << "Record was successfully added!" << endl;system("pause");

}

void EditStudent() {

int studentIdSearching;Block block;uint64\_t offset; system("cls");

if (!ShowDB()) { system("cls");cout << "Data Base is empty!" << endl;system("pause");return;}

cout << endl;cout << "Enter Student ID for editing:" << endl;cin >> studentIdSearching;system("cls");

int posOfModyfStud = SearchStudentInDB(studentIdSearching, block, offset);

if (posOfModyfStud == -1) {cout << "This Student ID is not found" << endl;system("pause");return;}

int userChoice;Block t\_blk;uint64\_t t\_off;while (true){

system("cls");cout << "Edit Student: "<<endl << endl;ShowZap(block.records[posOfModyfStud]);

cout << endl;cout << "What you want to edit:" << endl;cout << "1) Student ID" << endl;

cout << "2) Second name" << endl;cout << "3) First name" << endl;

cout << "4) Mid name" << endl; cout << "5) Group ID" << endl << endl;

cout << "0) Back" << endl << endl;cin >> userChoice;system("cls"); switch (userChoice){

case 1: int newId;cout << "Enter new stud ID:" << endl;cin >> newId;if (SearchStudentInDB(newId, t\_blk, t\_off) !=-1) {cout << "This Student ID already exists" << endl;system("pause");}

else {block.records[posOfModyfStud].studentId = newId;} break;

case 2:char newSecondName[30];cout << "Enter new second name:" << endl;cin >> newSecondName;

strcpy(block.records[posOfModyfStud].secondName, newSecondName);break;

case 3: char newFirstName[20]; cout << "Enter new first name:" << endl;cin >> newFirstName;

strcpy(block.records[posOfModyfStud].firstName, newFirstName); break;

case 4: char newmidName[30]; cout << "Enter new mid name:" << endl;cin >> newmidName;

strcpy(block.records[posOfModyfStud].midName, newmidName); break;

case 5: int newGroupID; cout << "Enter new group ID:" << endl;cin >> newGroupID;

block.records[posOfModyfStud].groupId = newGroupID; break;

case 0: fseek(file, offset, SEEK\_SET);fwrite(&block, sizeof(Block), 1, file); return;break;

default: break;

}

}

}

void DeleteStudent() {

int studentIdSearching; Block block; uint64\_t offset; system("cls");

if (!ShowDB()) { system("cls");cout << "Data Base is empty!" << endl;system("pause");return;}

cout << endl;cout << "Enter Student ID for deleting:" << endl;cin >> studentIdSearching;system("cls");

int posOfDelStud = SearchStudentInDB(studentIdSearching, block, offset);

if (posOfDelStud == -1) { cout << "This Student ID is not found" << endl;system("pause");return;}

Block lastBlock; uint64\_t lastOffset;fseek(file, -(long)sizeof(Block), SEEK\_END);lastOffset = ftell(file);

fread(&lastBlock, sizeof(Block), 1, file);

if (lastOffset == offset) {if (posOfDelStud == 4 || block.records[posOfDelStud + 1].studentId == -1) {

block.records[posOfDelStud] = Zap();

}

else {int i=1;while(posOfDelStud + i < 4 && block.records[posOfDelStud+i+1].studentId!=-1) { i++; }

block.records[posOfDelStud] = block.records[posOfDelStud + i];

block.records[posOfDelStud + i] = Zap();

}

lastBlock = block;

}else { int i = 0; while (lastBlock.records[i + 1].studentId != -1 && i < 4) { i++; }

block.records[posOfDelStud] = lastBlock.records[i]; lastBlock.records[i] = Zap();

fseek(file, lastOffset, SEEK\_SET); fwrite(&lastBlock, sizeof(Block), 1, file);

}

fseek(file, offset, SEEK\_SET); fwrite(&block, sizeof(Block), 1, file);

if (lastBlock.records[0].studentId == -1) { fclose(file);

HANDLE hFile = CreateFile(fileName.c\_str(), GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_RANDOM\_ACCESS, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {cout << "Can't delete block from file!" << endl;

system("pause"); exit(-1); }

SetFilePointer(hFile, lastOffset, 0, FILE\_BEGIN); SetEndOfFile(hFile);CloseHandle(hFile);

CreateFile();

}

}

int SearchStudentByAnyInfo(string &infoForSearching) {

Block block; int find = -1; fseek(file, 0, SEEK\_SET);

while (fread(&block, sizeof(Block), 1, file)) { for (int i = 0; i < 5; i++) {

if (to\_string(block.records[i].studentId) == infoForSearching || block.records[i].secondName == infoForSearching || block.records[i].firstName == infoForSearching || block.records[i].midName == infoForSearching|| to\_string(block.records[i].groupId) == infoForSearching) {

ShowZap(block.records[i]); find = 1;

}

}

} return find;

}

int main() {cout << "Enter filename:" << endl;cin >> fileName;CreateFile();string inInfo;int userChoice;

while (true) {

system("cls");

cout << "Main Menu: " << endl << endl; cout << "1) Add student" << endl;

cout << "2) Modify student" << endl; cout << "3) Delete student" << endl;

cout << "4) Find student" << endl; cout << "5) Show all students" << endl << endl;

cout << "0) Exit" << endl << endl; cin >> userChoice;

system("cls"); switch (userChoice){

case 1: AddStudentInDB(); break;

case 2: EditStudent(); break;

case 3: DeleteStudent(); break;

case 4: cout << "Enter information for searching:" << endl; cin >> inInfo; system("cls");

if (SearchStudentByAnyInfo(inInfo) == -1) {cout << "The entered information is incorrect or the student with such information doesn't exist!" << endl; }

system("pause"); break;

case 5:if (!ShowDB()) { system("cls");cout << "Data Base is empty!" << endl;}

system("pause"); break;

case 0: fclose(file); exit(0); break;

default: break;

}

}return 0;

}