МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ПРАКТИЧЕСКАЯ РАБОТА №2**

**«**Разработка сетевого интерфейса системы управления базой данных»

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Сычук Алексей Александрович*  «29» октября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Доцент кафедры ЗИ*  *Архипова Анастасия Борисовна*  «29» октября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы:** Реализовать сетевой интерфейс для СУБД из практики 1.

**Задание к работе:**

Требуется реализовать сетевой интерфейс для СУБД из практики 1.

При запуске приложения из первой практики СУБД должна ожидать соединение по протоколу tcp на порту 7432. При подключении требуется обработать запрос либо в отдельном потоке, либо в отдельном процессе, либо асинхронной задачей и отдать результат в ответ на запрос.

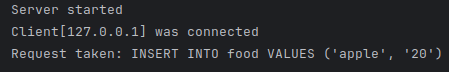
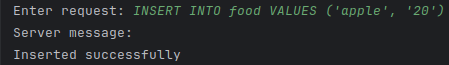
Также необходимо позаботится о блокировках на структуре БД если используются потоки или процессы.

**Методика выполнения работы:**

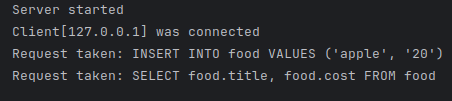
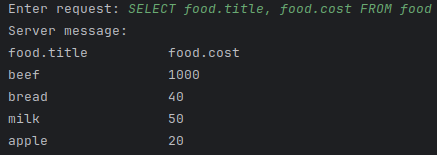
Запуск сервера и подключение клиента:



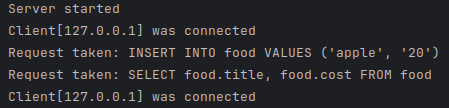
Вставка данных в бд:



Выборка данных из бд:



Подключение очередного клиента:



Отключение клиента:



**C++**

**Libs.h**

#ifndef LIBS\_H\_INCLUDED

#define LIBS\_H\_INCLUDED

#include <iostream>

#include <nlohmann/json.hpp>

#include <fstream>

#include <filesystem>

#include "dirent.h"

#include <string>

using namespace std;

#endif // LIBS\_H\_INCLUDED

**Structures.h**

#ifndef STRUCTURES\_H\_INCLUDED

#define STRUCTURES\_H\_INCLUDED

#include "libs.h"

enum class SQLRequest{

SELECT,

INSERT,

DELETE,

END,

UNKNOWN

};

template <typename T>

struct Node{

T data;

Node\* next;

Node(T val) : data(val), next(nullptr) {}

};

struct IntList{ // список чисел

Node<int>\* first;

Node<int>\* last;

IntList() : first(nullptr), last(nullptr) {}

bool is\_empty();

void push\_back(int);

Node<int>\* find(int);

void print(string);

void clear();

};

struct BoolList{ // список булевых элементов

Node<bool>\* first;

Node<bool>\* last;

BoolList() : first(nullptr), last(nullptr) {}

bool is\_empty();

void push\_back(bool);

void clear();

};

struct StringList{ // список строк

int listSize;

Node<string>\* first;

Node<string>\* last;

StringList() : listSize(0), first(nullptr), last(nullptr) {}

bool is\_empty();

void push\_back(string);

string print(string);

Node<string>\* find(int);

Node<string>\* word\_find(const string&);

string join(const char);

int index\_word(const string&);

void clear();

};

struct MatrixNode{

string data;

MatrixNode\* nextRow;

MatrixNode\* nextCol;

MatrixNode(string val) : data(val), nextRow(nullptr), nextCol(nullptr) {}

};

struct StringMatrix{ // матрица, заполняемая по стобцам

MatrixNode\* firstCol;

MatrixNode\* lastCol;

StringMatrix() : firstCol(nullptr), lastCol(nullptr){}

bool is\_empty();

void push\_right(string);

void push\_down(string, int);

string print();

void clear();

};

struct Array {

char\* data;

int size;

int capacity;

Array(int initialCapacity = 10) : size(0), capacity(initialCapacity){

data = new char[capacity];

}

~Array() {

delete[] data;

}

char\* get();

};

string operator\*(const string&, int);

#endif // STRUCTURES\_H\_INCLUDED

**Structures.cpp**

#include "structures.h"

bool IntList::is\_empty(){

return first == nullptr;

}

void IntList::push\_back(int data){ // добавление элемента в конец списка

Node<int>\* newElem = new Node(data);

if (is\_empty()){

first = newElem;

last = newElem;

return;

}

last->next = newElem;

last = newElem;

}

Node<int>\* IntList::find(int index){ // нахождение элемента по индексу

int counter = 0;

Node<int>\* current = first;

while (counter < index){

current = current->next;

++counter;

}

return current;

}

void IntList::print(string delimiter){ // вывод

if (is\_empty()) return;

Node<int>\* current = first;

while(current){

cout << current->data << delimiter;

current = current->next;

}

if (delimiter != "\n")

{

cout << endl;

}

}

void IntList::clear(){ // очистка

while (first != nullptr){

Node<int>\* temp = first;

first = first->next;

delete temp;

}

last = nullptr;

}

bool BoolList::is\_empty(){

return first == nullptr;

}

void BoolList::push\_back(bool data){ // добавление эоемента в конец списка

Node<bool>\* newElem = new Node(data);

if (is\_empty()){

first = newElem;

last = newElem;

return;

}

last->next = newElem;

last = newElem;

}

void BoolList::clear(){ // очистка

while (first != nullptr){

Node<bool>\* temp = first;

first = first->next;

delete temp;

}

last = nullptr;

}

bool StringList::is\_empty(){

return first == nullptr;

}

void StringList::push\_back(string data){ // добавление элемента в конец списка

++listSize;

Node<string>\* newElem = new Node(data);

if (is\_empty()){

first = newElem;

last = newElem;

return;

}

last->next = newElem;

last = newElem;

}

string StringList::print(string delimiter){ // вывод

if (is\_empty()) return "";

string toReturn;

Node<string>\* current = first;

while(current){

toReturn += current->data + delimiter;

current = current->next;

}

toReturn += "\n";

return toReturn;

}

Node<string>\* StringList::find(int index){ // нахождение элемента по индексу

if (index >= listSize || index < 0) cout << index << endl;

int counter = 0;

Node<string>\* current = first;

while (counter < index){

current = current->next;

++counter;

}

return current;

}

Node<string>\* StringList::word\_find(const string& word){ // нахождение элемента по значению

Node<string>\* current = first;

while(current->data != word){

current = current->next;

if (current->next == nullptr){break;}

}

return current;

}

string StringList::join(const char symbol){ // соеденение списка в строку

string joined;

Node<string>\* current = first;

while (current){

joined += current->data + symbol;

current = current->next;

}

return joined;

}

int StringList::index\_word(const string& word){ // получение индекса искомого слова

Node<string>\* current = first;

int id = 0;

while (current){

if (current->data == word){return id;}

++id;

current = current->next;

}

return -1;

}

void StringList::clear(){ // очистка

while (first != nullptr){

Node<string>\* temp = first;

first = first->next;

delete temp;

}

last = nullptr;

}

bool StringMatrix::is\_empty(){

return firstCol == nullptr;

}

void StringMatrix::push\_right(string text){ // добавление элемента в новый столбец

MatrixNode\* newElem = new MatrixNode(text);

if (is\_empty()){

firstCol = newElem;

lastCol = newElem;

return;

}

lastCol->nextCol = newElem;

lastCol = newElem;

}

void StringMatrix::push\_down(string text, int colNum){ // добавление элемента в новую строку в определенный столбец

MatrixNode\* newElem = new MatrixNode(text);

MatrixNode\* currCol = firstCol;

int cntr = 0;

while (cntr != colNum){

currCol = currCol->nextCol;

++cntr;

}

MatrixNode\* currRow = currCol;

while (currRow->nextRow != nullptr){

currRow = currRow->nextRow;

}

currRow->nextRow = newElem;

}

string StringMatrix::print(){ // вывод

StringList out;

for (auto col = firstCol; col != nullptr; col = col->nextCol){

int currRow = 0;

for (auto row = col; row != nullptr; row = row->nextRow){

if (col == firstCol){

out.push\_back(row->data + (string(" ") \* static\_cast<int>(20 - (row->data).size())));

}

else{

out.find(currRow)->data += row->data + (string(" ") \* static\_cast<int>(20 - (row->data).size()));

}

++currRow;

}

}

return out.print("\n");

}

void StringMatrix::clear(){ // очистка

while (firstCol != nullptr){

MatrixNode\* tempCol = firstCol;

firstCol = firstCol->nextCol;

while (tempCol->nextRow != nullptr){

MatrixNode\* tempRow = tempCol->nextRow;

tempCol->nextRow = tempRow->nextRow;

delete tempRow;

}

delete tempCol;

}

lastCol = nullptr;

}

char\* Array::get(){

return data;

}

string operator\*(const string& str, int n) { // переопределение операции умножения строки на число

string result;

for (int i = 0; i < n; ++i) {

result += str;

}

return result;

}

**Main.cpp**

#include "libs.h"

#include "structures.h"

#define \_BSD\_SOURCE 1

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <thread>

#include <mutex>

#include <atomic>

#define PORT 7432

int tuplesLim;

mutex mainMuter;

atomic<int> cntThreads(1);

const int MAX\_CLIENTS = 50;

StringList take\_section(StringList&, unsigned int, unsigned int);

void check\_active(const string&, const StringList&);

void make\_active(const string&, const StringList&);

void make\_inactive(const string&, const StringList&);

string create\_db();

StringList split(const string&, const string&);

string remove\_extra(string&);

void write\_in\_csv(const string&, StringList);

string insert\_into(const string&, StringList);

bool check\_filter\_delete(StringList&, StringList&, const string&);

string low\_id(const string&, int);

string delete\_from(const string&, StringList);

bool check\_filter\_select(const string&, const string&, int);

IntList cnt\_rows(StringMatrix&);

string select\_from(const string&, StringList);

SQLRequest get\_com (const string&);

string comp\_request(const string&, string);

void serve\_client(int, const string&);

void start\_server(const string&);

int main();

StringList take\_section(StringList& source, unsigned int frontInd, unsigned int backInd){ // взятие части списка

StringList out;

unsigned int currInd = frontInd;

while (currInd != backInd){

out.push\_back(source.find(currInd)->data);

++currInd;

}

return out;

}

void check\_active(const string& genPath, const StringList& tables){ // проверка, используется ли сейчас таблица

while (true){

bool isFree = true;

for (auto i = tables.first; i != nullptr; i = i->next){

string path = genPath + i->data + "/" + i->data + "\_lock.txt";

ifstream checkActive(path);

string line;

checkActive >> line;

checkActive.close();

if (line == "1"){

isFree = false;

}

}

if (isFree) {break;}

}

}

void make\_active(const string& genPath, const StringList& tables){ // занять таблицу

for (auto i = tables.first; i != nullptr; i = i->next){

string path = genPath + i->data + "/" + i->data + "\_lock.txt";

ofstream makeActive(path);

makeActive << 1;

makeActive.close();

}

}

void make\_inactive(const string& genPath, const StringList& tables){ // освободить таблицу

for (auto i = tables.first; i != nullptr; i = i->next){

string path = genPath + i->data + "/" + i->data + "\_lock.txt";

ofstream makeInactive(path);

makeInactive << 0;

makeInactive.close();

}

}

string create\_db(){ // создание базы данных, если ее нет

ifstream inFile("schema.json");

nlohmann::json schema;

inFile >> schema;

tuplesLim = (int)schema["tuples\_limit"];

string name = schema["name"];

if (filesystem::is\_directory(name)){

return (string)name;

}

filesystem::path basePath = name;

for (auto i : schema["structure"].items()){

filesystem::path dirPath = basePath / i.key();

filesystem::create\_directories(dirPath);

ofstream outfile(dirPath / "1.csv");

outfile << (i.key() + "\_pk;");

for (auto column = 0; column < i.value().size(); ++column){

if (column == i.value().size() - 1){

outfile << (string)i.value()[column] << endl;

}else{

outfile << (string)i.value()[column] << ";";

}

}

outfile.close();

ofstream lockfile(dirPath / (i.key() + "\_lock.txt")); // добавлЯяем файл блокировки

lockfile << "0";

lockfile.close();

ofstream keys(dirPath / (i.key() + "\_pk\_sequence.txt")); // добавляем счетчик id

keys << "1";

keys.close();

}

return (string)name;

}

StringList split(const string& str, const string& delimiter) { // разбиение строки в список

StringList result;

string currentPart;

int delimiterLength = delimiter.size();

for (auto i = 0; i < str.size(); ++i) {

int j = 0;

while (j < delimiterLength && i + j < str.size() && str[i + j] == delimiter[j]) {

++j;

}

if (j == delimiterLength) {

if (currentPart != "") {

result.push\_back(currentPart);

currentPart = "";

}

i += delimiterLength - 1;

} else {

currentPart += str[i];

}

}

if (!currentPart.empty()) {

result.push\_back(currentPart);

}

return result;

}

string remove\_extra(string& removeFrom){ // удаление лишних символов

string newStr;

for (auto i: removeFrom){

if (i == '(' || i == '\'' || i == ')' || i == ',' || i == ' '){

continue;

}

else {newStr += i;}

}

return newStr;

}

void write\_in\_csv(const string& path, StringList text){ // запись в csv файл

ofstream out(path, ios\_base::app);

for (int i = 0; i < text.listSize; ++i){

if(i != text.listSize - 1){

out << text.find(i)->data << ";";

}

else {

out << text.find(i)->data << endl;

}

}

}

string insert\_into(const string& schemaName, StringList command){ // вставка строки в бд

string table = command.find(2)->data; // получаем название таблицы

StringList tables;

tables.push\_back(table);

check\_active(schemaName + "/", tables);

make\_active(schemaName + "/", tables);

ifstream headerRead(schemaName + '/' + table + '/' + "1.csv"); // получение заголовка таблицы

string header;

headerRead >> header;

headerRead.close();

// обновление id

StringList data;

ifstream pkRead(schemaName + '/' + table + '/' + table + "\_pk\_sequence.txt");

string idStr;

getline(pkRead, idStr);

pkRead.close();

data.push\_back(idStr);

int newID = stoi(idStr) + 1;

for (int i = 4; i < command.listSize; ++i){

data.push\_back(remove\_extra(command.find(i)->data)); // чтение вставляемых данных

}

if (split(header, ";").listSize != data.listSize){

make\_inactive(schemaName + "/", tables);

tables.clear();

data.clear();

return "Wrong count of arguments\n";

}

ofstream pkWrite(schemaName + '/' + table + '/' + table + "\_pk\_sequence.txt");

pkWrite << to\_string(newID);

pkWrite.close();

// поиск свободного места

string path;

int currFile = 1;

do{

path = schemaName + '/' + table + '/';

ifstream check(path + to\_string(currFile) + ".csv");

if (check.bad()){

break;

}

int cntLines = -1;

string line;

while(check >> line){

++cntLines;

}

if (cntLines <= tuplesLim){ // если переполнение лимитов, то создаем новый файл

path += to\_string(currFile) + ".csv";

if (currFile > 1){

ofstream headerWrite(path);

headerWrite << header << endl;

headerWrite.close();

}

write\_in\_csv(path, data);

break;

}

++currFile;

}while(true);

make\_inactive(schemaName + "/", tables);

tables.clear();

data.clear();

return "Inserted successfully\n";

}

bool check\_filter\_delete(StringList& header, StringList& text, const string& filter){ // проверка фильтра для удаления

StringList orSplited = split(filter, " or ");

for (Node<string>\* i = orSplited.first; i != nullptr; i = i->next){ // слабый приоритет or

StringList andSplited = split(i->data, " and ");

bool isAnd = true;

for (Node<string>\* j = andSplited.first; j != nullptr; j = j->next){ // сильный приоритет and

StringList expression = split(j->data, " ");

string colName1 = split(expression.find(0)->data, ".").find(1)->data; // первая колонка

int colIndex1 = header.index\_word(colName1);

if (expression.find(2)->data[0] == '\''){ // если сравнение со строкой

if (text.find(colIndex1)->data != remove\_extra(expression.find(2)->data)){

isAnd = false;

break;

}

}

else { // если сравнение двух элементов таблицы

string colName2 = split(expression.find(2)->data, ".").find(1)->data;

int colIndex2 = header.index\_word(colName2);

if (text.find(colIndex1)->data != text.find(colIndex2)->data){

isAnd = false;

break;

}

}

}

if (isAnd){orSplited.clear(); return true;}

andSplited.clear();

}

orSplited.clear();

return false;

}

string low\_id(const string& command, int lowOn){ // уменьшить id после удаления элементов

StringList splited = split(command, ";");

int id = stoi(splited.find(0)->data);

id -= lowOn;

splited.find(0)->data = to\_string(id);

string newCommand = splited.join(';');

splited.clear();

return newCommand;

}

string delete\_from(const string& schemaName, StringList command){ // основная функция удаление

if (command.listSize < 3)

{

return "Wrong count of arguments\n";

}

StringList tables; // получение таблиц

tables.push\_back(command.find(2)->data);

check\_active(schemaName + "/", tables);

make\_active(schemaName + "/", tables);

string path = schemaName + '/' + command.find(2)->data + '/';

int currentFile = 1;

if (command.word\_find("WHERE") == command.last){ // если есть фильтр, то пересоздаем таблицу

while (remove((path + to\_string(currentFile) + ".csv").c\_str())){

++currentFile;

}

ifstream inFile("schema.json");

nlohmann::json schema;

inFile >> schema;

StringList columns;

columns.push\_back(command.find(2)->data + "\_pk");

for (auto i: schema["structure"][command.find(2)->data]){

columns.push\_back(i);

}

ofstream newFirst(path + "1.csv");

newFirst.close();

write\_in\_csv(path + "1.csv", columns);

ofstream updateId(path + command.find(2)->data + "\_pk\_seqquence.txt");

updateId << "1";

updateId.close();

tables.clear();

columns.clear();

return "Deleted successfully\n";

}

StringList filter = take\_section(command, 4, command.listSize); // получение фильтра

string toSplit = filter.join(' ');

int diffId = 0;

do{ // проверяем каждую строчку на фильтр

ifstream readFile(path + to\_string(currentFile) + ".csv");

if (!readFile.is\_open()){

break;

}

string strHeader;

readFile >> strHeader;

StringList header = split(strHeader, ";");

string line;

StringList save;

while(readFile >> line){

StringList data = split(line, ";");

if (!check\_filter\_delete(header, data, toSplit)){

string temp = low\_id(line, diffId);

save.push\_back(temp); // сохраняем, если подойдет

}

else {

++diffId; // разница в id возрастает

}

}

readFile.close();

ofstream writeFile(path + to\_string(currentFile) + ".csv");

writeFile << strHeader << endl;

for (Node<string>\* i = save.first; i != nullptr; i = i->next){ // перезапись файла

writeFile << i->data << endl;

}

writeFile.close();

++currentFile;

header.clear();

save.clear();

}while(true);

// изменение max id

ifstream pkRead(schemaName + '/' + command.find(2)->data + '/' + command.find(2)->data + "\_pk\_sequence.txt");

string idStr;

getline(pkRead, idStr);

pkRead.close();

int newID = stoi(idStr) - diffId;

ofstream pkWrite(schemaName + '/' + command.find(2)->data + '/' + command.find(2)->data + "\_pk\_sequence.txt");

pkWrite << newID;

pkWrite.close();

make\_inactive(schemaName + "/", tables);

tables.clear();

filter.clear();

return "Deleted successfully\n";

}

// проверка условия для select

bool check\_filter\_select(const string& schemaName, const string& filter, int currStr){

Node<string>\* orSplited = split(filter, " OR ").first;

while(orSplited != nullptr){

Node<string>\* andSplited = split(orSplited->data, " AND ").first;

bool isAnd = true;

while(andSplited != nullptr){

StringList eqlSplited = split(andSplited->data, " = ");

StringList leftSplited = split(eqlSplited.first->data, ".");

string leftTab = leftSplited.first->data, leftCol = leftSplited.first->next->data;

int currFile = 1;

int currLine = 0;

string leftHeader;

string leftLine;

while (true){

ifstream leftRead(schemaName + "/" + leftTab + "/" + to\_string(currFile) + ".csv");

if (!leftRead.is\_open()){

break;

}

leftRead >> leftHeader;

while(leftRead >> leftLine && currLine != currStr){

++currLine;

}

if (currLine == currStr) break;

++currFile;

}

StringList splitedLeftHeader = split(leftHeader, ";");

int leftColIndex = splitedLeftHeader.index\_word(leftCol);

string leftValue = split(leftLine, ";").find(leftColIndex)->data;

if ((eqlSplited.first->next->data)[0] == '\''){

string rightValue = remove\_extra(eqlSplited.first->next->data);

if (leftValue != rightValue){

isAnd = false;

break;

}

}

else {

StringList rightSplited = split(eqlSplited.first->next->data, ".");

string rightTab = rightSplited.first->data, rightCol = remove\_extra(rightSplited.first->next->data);

currFile = 1;

currLine = 0;

string rightHeader;

string rightLine;

while (true){

ifstream rightRead(schemaName + "/" + rightTab + "/" + to\_string(currFile) + ".csv");

if (!rightRead.is\_open()){

break;

}

rightRead >> rightHeader;

while(rightRead >> rightLine && currLine != currStr){

++currLine;

}

if (currLine == currStr) break;

++currFile;

}

StringList splitedRightHeader = split(rightHeader, ";");

int rightColIndex = splitedRightHeader.index\_word(rightCol);

string rightValue = split(rightLine, ";").find(rightColIndex)->data;

if (leftValue != rightValue){

isAnd = false;

break;

}

}

andSplited = andSplited->next;

}

if (isAnd){

return true;

}

orSplited = orSplited->next;

}

return false;

}

IntList cnt\_rows(StringMatrix& matrix){ // подсчет количества рядов в каждом столбце

IntList eachCol;

for (auto i = matrix.firstCol; i != nullptr; i = i->nextCol){

int cntRow = 0;

for (auto j = i->nextRow; j != nullptr; j = j->nextRow){

++cntRow;

}

eachCol.push\_back(cntRow);

}

return eachCol;

}

string select\_from(const string& schemaName, StringList command){ // функция получения выборки

string genPath = schemaName + '/';

int whereIndex = command.index\_word("WHERE");

whereIndex = whereIndex > 0 ? whereIndex : command.listSize;

// получаем таблицы

StringList tables = take\_section(command, command.index\_word("FROM") + 1, whereIndex);

for (auto i = tables.first; i != nullptr; i = i->next){

i->data = remove\_extra(i->data);

}

check\_active(genPath, tables);

make\_active(genPath, tables);

// получаем колонки

StringList columns = take\_section(command, command.index\_word("SELECT") + 1, command.index\_word("FROM"));

for (auto i = columns.first; i != nullptr; i = i->next){

i->data = remove\_extra(i->data);

}

StringMatrix toOut;

int currTable = 0;

int currCol = 0;

if (command.word\_find("WHERE") == command.last){ // если нет фильтра

int totalCnt = 1;

IntList strInTable;

for (auto i = tables.first; i != nullptr; i = i->next){

int currFile = 1;

int cntLines = 0;

string path = genPath + i->data + '/';

// подсчет строк в каждом файле таблиц

do{

ifstream check(path + to\_string(currFile) + ".csv");

if (!check.is\_open()){

break;

}

string line;

while(check >> line){

++cntLines;

}

++currFile;

}while(true);

strInTable.push\_back(--cntLines);

totalCnt \*= cntLines; // получаем число повторений для crossjoin

}

// заполнение матрицы для конечного вывода

for (auto i = tables.first; i != nullptr; i = i->next){

string path = genPath + i->data + '/';

if (strInTable.find(currTable)->data != 0){

totalCnt /= strInTable.find(currTable)->data;

}

for (auto j = columns.first; j != nullptr; j = j->next){

StringList tabNCol = split(j->data, ".");

if (tabNCol.find(0)->data != i->data){

continue;

}

ifstream forHead(path + "1.csv");

string strHeader;

forHead >> strHeader;

forHead.close();

StringList header = split(strHeader, ";");

int takenId = header.index\_word(tabNCol.find(1)->data);

toOut.push\_right(tables.find(currTable)->data + "." + tabNCol.find(1)->data);

int currFile = 1;

do{

ifstream readFile(path + to\_string(currFile) + ".csv");

if (!readFile.is\_open()){

break;

}

string line;

readFile >> line;

while(readFile >> line){

StringList splited = split(line, ";");

for (int k = 0; k < totalCnt; ++k){

toOut.push\_down(splited.find(takenId)->data, currCol);

}

splited.clear();

}

readFile.close();

if (currTable != 0){

MatrixNode\* currHead = toOut.lastCol;

int cntrCurr = 0;

for (auto k = currHead->nextRow; k != nullptr; k = k->nextRow){

++cntrCurr;

}

int cntrFirst = 0;

for (auto k = toOut.firstCol; k != nullptr; k = k->nextRow){

++cntrFirst;

}

if (cntrCurr == 0){++currFile; continue;}

// дублирование столбца вниз нужное количество раз

for (auto m = 0; m < (cntrFirst / cntrCurr) - 1; ++m){

MatrixNode\* currRow = currHead->nextRow;

for (int k = 0; k < cntrCurr; ++k){

toOut.push\_down(currRow->data, currCol);

currRow = currRow->nextRow;

}

}

}

++currFile;

}while(true);

++currCol;

}

++currTable;

}

// проверка наличия каких-то данных в полученной матрице

IntList eachCol = cnt\_rows(toOut);

for (auto i = eachCol.first; i != nullptr; i = i->next)

{

if (i->data == 0){

string toReturn;

toReturn += string("-") \* 40 + "\n";

for (auto j = toOut.firstCol; j != nullptr; j = j->nextCol){

toReturn += j->data + " ";

}

toReturn += "\n" + string("-") \* 40 + "\n";

return toReturn;

}

}

string toReturn = toOut.print();

make\_inactive(genPath, tables);

tables.clear();

columns.clear();

strInTable.clear();

toOut.clear();

eachCol.clear();

return toReturn;

}

// получение фильтра

string filter = take\_section(command, command.index\_word("WHERE") + 1, command.listSize).join(' ');

// аналогичный проход по файлам с проверкой условий

for (auto i = tables.first; i != nullptr; i = i->next){

string path = genPath + i->data + "/";

for (auto j = columns.first; j != nullptr; j = j->next){

StringList tabNCol = split(j->data, ".");

if (tabNCol.find(0)->data != i->data){

continue;

}

ifstream forHead(path + "1.csv");

string strHeader;

forHead >> strHeader;

forHead.close();

StringList header = split(strHeader, ";");

int takenId = header.index\_word(tabNCol.find(1)->data);

int currFile = 1;

int currStr = 0;

toOut.push\_right(j->data);

do{

ifstream readFile(path + to\_string(currFile) + ".csv");

if (!readFile.is\_open()){

break;

}

string line;

readFile >> line;

while(readFile >> line){

StringList splited = split(line, ";");

if (check\_filter\_select(schemaName, filter, currStr)){

toOut.push\_down(splited.find(takenId)->data, currCol);

}

++currStr;

}

readFile.close();

++currFile;

}while(true);

++currCol;

}

}

IntList cntInEach = cnt\_rows(toOut);

for (auto i = cntInEach.first; i != nullptr; i = i->next){

if (i->data == 0){

string toReturn;

toReturn += string("-") \* 40 + "\n";

for (auto j = toOut.firstCol; j != nullptr; j = j->nextCol){

toReturn += j->data + " ";

}

toReturn += "\n" + string("-") \* 40 + "\n";

return toReturn;

}

}

int total = 1;

for (auto i = cntInEach.first->next; i != nullptr; i = i->next){

total \*= i->data;

}

// повторяем каждую строчку нужное кол-во раз

StringMatrix temp;

currCol = 0;

for (auto i = toOut.firstCol; i != nullptr; i = i->nextCol){

temp.push\_right(i->data);

for (auto j = i->nextRow; j != nullptr; j = j->nextRow){

for (int k = 0; k < total; ++k){

temp.push\_down(j->data, currCol);

}

}

total /= cntInEach.find(currCol)->data;

++currCol;

}

// повторяем каждый блок нужное кол-во раз

toOut.clear();

StringMatrix finalOut;

cntInEach = cnt\_rows(temp);

currCol = 0;

for (auto i = temp.firstCol; i != nullptr; i = i->nextCol){

finalOut.push\_right(i->data);

for (int k = 0; k < (cntInEach.find(0)->data / cntInEach.find(currCol)->data); ++k){

for (auto j = i->nextRow; j != nullptr; j = j->nextRow){

finalOut.push\_down(j->data, currCol);

}

}

++currCol;

}

string toReturn = finalOut.print();

temp.clear();

finalOut.clear();

cntInEach.clear();

make\_inactive(genPath, tables);

tables.clear();

columns.clear();

return toReturn;

}

SQLRequest get\_com (const string& command){ // выбор токена

if (command == "SELECT") {return SQLRequest::SELECT;}

if (command == "INSERT") {return SQLRequest::INSERT;}

if (command == "DELETE") {return SQLRequest::DELETE;}

return SQLRequest::UNKNOWN;

}

string comp\_request(const string& schemaName, string request){

StringList splited = split(request, " "); // делим запрос

SQLRequest choice = get\_com(splited.find(0)->data); // полчаем токен

switch (choice){ // в зависимости от токена вызываем нужную функцию

case SQLRequest::SELECT: return select\_from(schemaName, splited);

case SQLRequest::INSERT: return insert\_into(schemaName, splited);

case SQLRequest::DELETE: return delete\_from(schemaName, splited);

case SQLRequest::UNKNOWN: return "Wrong command!";

}

}

void serve\_client(int servSocket, const string& schemaName){

++cntThreads; // увеличиваем количество клиентов на сервере

while (true) { // начинаем слушать запросы

Array client(1024);

memset(client.get(), 0, client.size); // очищаем буфер

ssize\_t bytesRead = recv(servSocket, client.get(), client.size - 1, 0); // получаем запрос

if (bytesRead <= 0) { // либо клиент отключился, либо произошла ошибкаа при передаче данных

cout << "Error or client was disconnected" << endl;

break;

}

{

lock\_guard<mutex> lock(mainMuter); // ограничиваем доступ, чтобы не было ошибок в выводе

client.get()[bytesRead] = '\0';

cout << "Request taken: " << client.get() << endl;

}

string answer = "Server message:\n";

string request = client.get();

string partRes = comp\_request(schemaName, request); // отправляем запрос на выполнение

answer += partRes; // формируем ответ

send(servSocket, answer.c\_str(), answer.size(), 0); // отправляем ответ

}

close(servSocket); // закрываем сокет для клиента

--cntThreads; // уменьшаем количество клиентов

}

void start\_server(const string& schemaName) {

int serverSocket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

// создание сокета

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

cerr << "Error of create socket" << endl;

return;

}

// настройка параметров сокета

if (setsockopt(serverSocket, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt)) == -1) {

cerr << "Error of setting parameters of socket" << endl;

return;

}

address.sin\_family = AF\_INET; // IPv4

address.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // установка IP

address.sin\_port = htons(PORT); // уановка порта

// привязкаа сокета к адресу

if (bind(serverSocket, (struct sockaddr \*)&address, sizeof(address)) < 0) {

cerr << "Error of binding" << endl;

return;

}

// прослушивание подключений

if (listen(serverSocket, 1) < 0) {

cerr << "Error of socket listening" << endl;

return;

}

cout << "Server started" << endl;

while (true){ // начинаем прослушивание

sockaddr\_in clientAddress;

socklen\_t clientSize = sizeof(clientAddress);

int clientSocket = accept(serverSocket, (struct sockaddr\*)&clientAddress, &clientSize); // принимаем подключение клиента

if(clientSocket < 0){

cout << "Error to connect client" << endl;

continue;

}

if(cntThreads <= MAX\_CLIENTS){ // если на сервере есть место

char\* clientIP = inet\_ntoa(clientAddress.sin\_addr); // получаем IP клиента

cout << "Client[" << clientIP << "] was connected" << endl; // выводим клиента, который подключился

thread(serve\_client, clientSocket, schemaName).detach(); // выводим клиента в другой поток

// и отключаем отслеживание

string answer = "Successfully connected to the server";

send(clientSocket, answer.c\_str(), answer.size(), 0); // отправляем клиенту успешное подключение

}

else{

string answer = "A lot of clients now, try it later";

send(clientSocket, answer.c\_str(), answer.size(), 0);

close(clientSocket);

}

}

close(serverSocket); // закрываем сервер

}

int main()

{

string schemaName = create\_db(); // создаем бд из json, если ее не было

start\_server(schemaName); // запускаем сервер

return 0;

}

**Client.cpp**

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 7432

using namespace std;

struct Array {

char\* data;

int size;

int capacity;

Array(int initialCapacity = 10) : size(0), capacity(initialCapacity){

data = new char[capacity];

}

~Array() {

delete[] data;

}

char\* get(){

return data;

}

};

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

cerr << "Error to create socket" << endl;

return -1;

}

// настройка адреса сервера

serv\_addr.sin\_family = AF\_INET; // IPv4

serv\_addr.sin\_port = htons(PORT); // установка порта

// преобразование ip в бинарный формат

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

cerr << "Incorrect address" << endl;

return -1;

}

// подключение к серверу

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

cerr << "Connection error" << endl;

return -1;

}

char buffer[1024]; // буфер для получаемого сообзения

int connectResp = read(sock, buffer, 1024); // ACK на подключение

cout << "Server message: " << buffer << endl;

while (true)

{

string request;

cout << "Enter request: ";

getline(cin, request);

if (request == "end")

{

break;

}

send(sock, request.c\_str(), request.size(), 0); // отправляем запрос на сервер

Array response(1024);

memset(response.data, 0, response.size); // очищаем область памяти массива

ssize\_t byteResp = recv(sock, response.get(), response.size - 1, 0); // получение ответа

if (byteResp > 0)

{

response.get()[byteResp] = '\0';

cout << response.get();

}

else // в случае, если возникла ошибка заккрываем сокет

{

cout << "Getting packages error" << endl;

close(sock);

cout << "You have been forcibly disconnected" << endl;

return 0;

}

}

close(sock);

cout << "You were successfully disconnected"; // отключаемся от сервера

return 0;

}

**Вывод**

В ходе выполнения данной практической работы были получены навыки работы с локальными сетями в с++, а также использовании tcp протокола для обмена данными между клиентом и сервером.