МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ОТЧЁТ**

**по практической работе №1**

**«**Разработка СУБД**»**

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

Выполнил:Проверил:

Студент гр. «АБс-324», «АВТФ» *доцент кафедры ЗИ*

*Клаус В. А. Архипова А. Б.*

«\_\_» \_\_\_\_\_\_ 2024 г.«\_\_\_» \_\_\_\_\_\_ 2024 г.

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

(подпись) (подпись)

Новосибирск 2024

**Задание:** Требуется реализовать реляционную СУБД с поддержкой запросов на языке SQL. СУБД использует единственный тип данных - строка.

Требуется поддержать следующие SQL выражения:

SELECT <> FROM <> - выборка

WHERE и операторы OR , AND - фильтрация

INSERT INTO - вставка данных в таблицы

DELETE FROM - удаление данных из таблицы

# Модули программы

## schema.json:

{

"names": "БдАвтосалона",

"tuples\_limit": 5,

"structure": {

"cars": ["марка", "год", "пробег", "цена"],

"users": ["фамилия", "имя", "марка"]

}

}

## headers:

include.h

#pragma once

using namespace std;

#include <iostream>

#include <fstream>

#include <sstream>

list.h

#pragma once

#include "includes.h"

template<typename T>

struct Node {

T data;

Node\* next;

Node(T value); // конструктор узла

};

template<typename T>

struct SinglyLinkedList {

Node<T>\* head;

int size = 0;

SinglyLinkedList(); // конструктор листа

~SinglyLinkedList(); // деструктор

void print(); // вывод списка

void push\_front(T value); // добавление элемента в начало

void push\_back(T value); // добавление элемента в конец

void pop\_front(); // удаление элемента с начала

void pop\_back(); // удаление элемента с конца

void remove(T value); // удаление элемента по значению

void replace(int index, T newValue); // ф-ия замены элемента по индексу

int getindex(T value); // поиск элемента по значению

T getvalue(int index); // поиск элемента по индексу

};

template <typename T>

struct Node2 {

T data;

Node2\* next;

Node2\* prev;

Node2(T value); // конструктор узла

};

template <typename T>

struct DoublyLinkedList {

Node2<T>\* head;

Node2<T>\* tail;

int size = 0;

DoublyLinkedList(); // конструктор листа

~DoublyLinkedList(); // деструктор

void print(); // вывод списка

void push\_front(T value); // добавление элемента в начало

void push\_back(T value); // добавление элемента в конец

void pop\_front(); // удаление элемента в начале

void pop\_back(); // удаление элемента в конце

void remove(T value); // удаление элемента по значению

bool find(T value); // поиска элемента по значению

};

#include "../src/list.cpp" // Включаем реализацию шаблона

## 

## 

## файлы cpp:

list.cpp

#include "../include/list.h"

template<typename T>

Node<T>::Node(T value) : data(value), next(nullptr) {}

template<typename T>

SinglyLinkedList<T>::SinglyLinkedList() : head(nullptr) {}

template<typename T>

void SinglyLinkedList<T>::print() {

Node<T>\* current = head;

while (current) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

template<typename T>

void SinglyLinkedList<T>::push\_front(T value) {

Node<T>\* newNode = new Node<T>(value);

newNode->next = head;

head = newNode;

size++;

}

template<typename T>

void SinglyLinkedList<T>::push\_back(T value) {

Node<T>\* newNode = new Node<T>(value);

if (head == nullptr) {

head = newNode;

} else {

Node<T>\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

size++;

}

template<typename T>

void SinglyLinkedList<T>::pop\_front() {

if (head == nullptr) return;

Node<T>\* temp = head;

head = head->next;

delete temp;

size--;

}

template<typename T>

void SinglyLinkedList<T>::pop\_back() {

if (head == nullptr) return;

if (!head->next) { // если только 1 элемент

delete head;

head = nullptr;

return;

}

Node<T>\* current = head;

while (current->next && current->next->next) {

current = current->next;

}

delete current->next; // Удаляем последний элемент

current->next = nullptr; // Обнуляем указатель

size--;

}

template<typename T>

void SinglyLinkedList<T>::remove(T value) {

if (head == nullptr) return;

if (head->data == value) {

pop\_front();

return;

}

Node<T>\* current = head;

while (current->next) {

if (current->next->data == value) {

Node<T>\* temp = current->next;

current->next = current->next->next;

delete temp;

return;

}

current = current->next;

}

size--;

}

template<typename T>

void SinglyLinkedList<T>::replace(int index, T newValue) {

if (index < 0 || index >= size) {

cout << "Index out of bounds." << endl;

return;

}

Node<T>\* current = head;

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

current = current->next;

}

current->data = newValue;

}

template<typename T>

int SinglyLinkedList<T>::getindex(T value) {

Node<T>\* current = head;

int index = 0;

while (current) {

if (current->data == value) {

return index; // Элемент найден, возвращаем индекс

}

current = current->next;

index++;

}

return -1; // Если элемент не найден, возвращаем -1

}

template<typename T>

T SinglyLinkedList<T>::getvalue(int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range");

}

Node<T>\* current = head;

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

current = current->next;

}

return current->data;

}

template<typename T>

SinglyLinkedList<T>::~SinglyLinkedList() {

while (head) {

pop\_front();

}

}

template<typename T>

Node2<T>::Node2(T value) : data(value), next(nullptr), prev(nullptr) {}

template<typename T>

DoublyLinkedList<T>::DoublyLinkedList() : head(nullptr), tail(nullptr) {}

template<typename T>

void DoublyLinkedList<T>::print() {

Node2<T>\* current = head;

while (current) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

template<typename T>

void DoublyLinkedList<T>::push\_front(T value) {

Node2<T>\* newNode = new Node<T>(value);

if (head == nullptr) {

head = tail = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

size++;

}

template<typename T>

void DoublyLinkedList<T>::push\_back(T value) {

Node2<T>\* newNode = new Node<T>(value);

if (tail == nullptr) {

head = tail = newNode;

} else {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

size++;

}

template<typename T>

void DoublyLinkedList<T>::pop\_front() {

if (head == nullptr) return;

Node2<T>\* temp = head;

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

} else {

tail = nullptr; // Список стал пустым

}

delete temp;

size--;

}

template<typename T>

void DoublyLinkedList<T>::pop\_back() {

if (tail == nullptr) return; // Список пуст

Node2<T>\* temp = tail;

tail = tail->prev;

if (tail != nullptr) {

tail->next = nullptr;

} else {

head = nullptr; // Список стал пустым

}

delete temp;

size--;

}

template<typename T>

void DoublyLinkedList<T>::remove(T value) {

Node2<T>\* current = head;

while (current) {

if (current->data == value) {

if (current->prev) {

current->prev->next = current->next;

} else {

head = current->next; // Удаляем голову

}

if (current->next) {

current->next->prev = current->prev;

} else {

tail = current->prev; // Удаляем хвост

}

delete current;

return; // Выход после удаления первого найденного элемента

}

current = current->next;

}

size--;

}

template<typename T>

bool DoublyLinkedList<T>::find(T value) {

Node2<T>\* current = head;

while (current) {

if (current->data == value) {

return true; // Элемент найден

}

current = current->next;

}

return false; // Элемент не найден

}

template<typename T>

DoublyLinkedList<T>::~DoublyLinkedList() {

while (head) {

pop\_front();

}

}

## main cpp:

#include "../include/json.hpp"

#include "../include/list.h"

int CountLine(string& filepath) { // ф-ия подсчёта строк в файле

ifstream file;

file.open(filepath);

int countline = 0;

string line;

while(getline(file, line)) {

countline++;

}

file.close();

return countline;

}

string finput(string& filepath) { // чтение из файла

string result, str;

ifstream fileinput;

fileinput.open(filepath);

while (getline(fileinput, str)) {

result += str + '\n';

}

result.pop\_back();

fileinput.close();

return result;

}

void foutput(string& filepath, string text) { // запись в файл

ofstream fileoutput;

fileoutput.open(filepath);

fileoutput << text;

fileoutput.close();

}

struct DataBase {

string nameBD; // название БД

int tupleslimit; // лимит строк

SinglyLinkedList<string> nametables; // названия таблиц

SinglyLinkedList<string> stlb; // столбцы таблиц

SinglyLinkedList<int> fileindex; // кол-во файлов таблиц

SinglyLinkedList<int> countlines; // кол-во строк таблиц

struct Where { // структура для фильтрации

string table;

string column;

string value;

string logicalOP;

bool check; // В частности для select, проверка условия(если просто условие - true, если условиестолбец - false)

};

void parse() { // ф-ия парсинга

nlohmann::json objJson;

ifstream fileinput;

fileinput.open("../schema.json");

fileinput >> objJson;

fileinput.close();

if (objJson["names"].is\_string()) {

nameBD = objJson["names"]; // Парсим каталог

} else {

cout << "Объект каталога не найден!" << endl;

exit(0);

}

tupleslimit = objJson["tuples\_limit"];

// парсим подкаталоги

if (objJson.contains("structure") && objJson["structure"].is\_object()) { // проверяем, существование объекта и является ли он объектом

for (auto elem : objJson["structure"].items()) {

nametables.push\_back(elem.key());

string kolonki = elem.key() + "\_pk\_sequence,"; // добавление первичного ключа

for (auto str : objJson["structure"][elem.key()].items()) {

kolonki += str.value();

kolonki += ',';

}

kolonki.pop\_back(); // удаление последней запятой

stlb.push\_back(kolonki);

fileindex.push\_back(1);

countlines.push\_back(1);

}

} else {

cout << "Объект подкаталогов не найден!" << endl;

exit(0);

}

}

void mkdir() { // ф-ия формирования директории

string command;

command = "mkdir ../" + nameBD; // каталог

system(command.c\_str());

for (int i = 0; i < nametables.size; ++i) { // подкаталоги и файлы в них

command = "mkdir ../" + nameBD + "/" + nametables.getvalue(i);

system(command.c\_str());

string filepath = "../" + nameBD + "/" + nametables.getvalue(i) + "/1.csv";

ofstream file;

file.open(filepath);

file << stlb.getvalue(i) << endl;

file.close();

// Блокировка таблицы

filepath = "../" + nameBD + "/" + nametables.getvalue(i) + "/" + nametables.getvalue(i) + "\_lock.txt";

file.open(filepath);

file << "open";

file.close();

}

}

void checkcommand(string& command) { // ф-ия фильтрации команд

if (command.substr(0, 11) == "insert into") {

command.erase(0, 12);

isValidInsert(command);

} else if (command.substr(0, 11) == "delete from") {

command.erase(0, 12);

isValidDel(command);

} else if (command.substr(0, 6) == "select") {

command.erase(0, 7);

isValidSelect(command);

} else if (command == "exit") {

exit(0);

} else cout << "Ошибка, неизвестная команда!" << endl;

}

// ф-ии делита

void isValidDel(string& command) { // ф-ия обработки команды DELETE

string table, conditions;

int position = command.find\_first\_of(' ');

if (position != -1) {

table = command.substr(0, position);

conditions = command.substr(position + 1);

} else table = command;

if (nametables.getindex(table) != -1) { // проверка таблицы

if (conditions.empty()) { // если нет условий, удаляем все

del(table);

} else {

if (conditions.substr(0, 6) == "where ") { // проверка наличия where

conditions.erase(0, 6);

SinglyLinkedList<Where> cond;

Where where;

position = conditions.find\_first\_of(' '); ////

if (position != -1) { // проверка синтаксиса

where.column = conditions.substr(0, position);

conditions.erase(0, position+1);

int index = nametables.getindex(table);

string str = stlb.getvalue(index);

stringstream ss(str);

bool check = false;

while (getline(ss, str, ',')) if (str == where.column) check = true;

if (check) { // проверка столбца

if (conditions[0] == '=' && conditions[1] == ' ') { // проверка синтаксиса

conditions.erase(0, 2);

position = conditions.find\_first\_of(' ');

if (position == -1) { // если нет лог. оператора

where.value = conditions;

delWithValue(table, where.column, where.value);

} else { // если есть логический оператор

where.value = conditions.substr(0, position);

conditions.erase(0, position+1);

cond.push\_back(where);

position = conditions.find\_first\_of(' ');

if ((position != -1) && (conditions.substr(0, 2) == "or" || conditions.substr(0, 3) == "and")) {

where.logicalOP = conditions.substr(0, position);

conditions.erase(0, position + 1);

position = conditions.find\_first\_of(' ');

if (position != -1) {

where.column = conditions.substr(0, position);

conditions.erase(0, position+1);

index = nametables.getindex(table);

str = stlb.getvalue(index);

stringstream iss(str);

bool check = false;

while (getline(iss, str, ',')) if (str == where.column) check = true;

if (check) { // проверка столбца

if (conditions[0] == '=' && conditions[1] == ' ') { // проверка синтаксиса

conditions.erase(0, 2);

position = conditions.find\_first\_of(' ');

if (position == -1) {

where.value = conditions;

cond.push\_back(where);

delWithLogic(cond, table);

} else cout << "Ошибка, нарушен синтаксис команды4!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды3!" << endl;

} else cout << "Ошибка, нет такого столбца!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды2!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды1!" << endl;

}

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нет такого столбца!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!"<< endl;

}

} else cout << "Ошибка, нет такой таблицы!" << endl;

}

void del(string& table) { // ф-ия удаления всех строк таблицы

string filepath;

int index = nametables.getindex(table);

if (checkLockTable(table)) {

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "close");

// очищаем все файлы

int copy = fileindex.getvalue(index);

while (copy != 0) {

filepath = "../" + nameBD + "/" + table + "/" + to\_string(copy) + ".csv";

foutput(filepath, "");

copy--;

}

foutput(filepath, stlb.getvalue(index)+"\n"); // добавляем столбцы в 1.csv

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "open");

cout << "Команда выполнена!" << endl;

} else cout << "Ошибка, таблица используется другим пользователем!" << endl;

}

void delWithValue(string& table, string& stolbec, string& values) { // ф-ия удаления строк таблицы по значению

string filepath;

int index = nametables.getindex(table);

if (checkLockTable(table)) {

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "close");

// нахождение индекса столбца в файле

string str = stlb.getvalue(index);

stringstream ss(str);

int stolbecindex = 0;

while (getline(ss, str, ',')) {

if (str == stolbec) break;

stolbecindex++;

}

// удаление строк

int copy = fileindex.getvalue(index);

while (copy != 0) {

filepath = "../" + nameBD + "/" + table + "/" + to\_string(copy) + ".csv";

string text = finput(filepath);

stringstream stroka(text);

string filteredlines;

while (getline(stroka, text)) {

stringstream iss(text);

string token;

int currentIndex = 0;

bool shouldRemove = false;

while (getline(iss, token, ',')) {

if (currentIndex == stolbecindex && token == values) {

shouldRemove = true;

break;

}

currentIndex++;

}

if (!shouldRemove) filteredlines += text + "\n";

}

foutput(filepath, filteredlines);

copy--;

}

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "open");

cout << "Команда выполнена!" << endl;

} else cout << "Ошибка, таблица используется другим пользователем!" << endl;

}

void delWithLogic(SinglyLinkedList<Where>& conditions, string& table) { // ф-ия удаления строк таблицы с логикой

string filepath;

int index = nametables.getindex(table);

if (checkLockTable(table)) {

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "close");

// нахождение индекса столбцов в файле

SinglyLinkedList<int> stlbindex;

for (int i = 0; i < conditions.size; ++i) {

string str = stlb.getvalue(index);

stringstream ss(str);

int stolbecindex = 0;

while (getline(ss, str, ',')) {

if (str == conditions.getvalue(i).column) {

stlbindex.push\_back(stolbecindex);

break;

}

stolbecindex++;

}

}

// удаление строк

int copy = fileindex.getvalue(index);

while (copy != 0) {

filepath = "../" + nameBD + "/" + table + "/" + to\_string(copy) + ".csv";

string text = finput(filepath);

stringstream stroka(text);

string filteredRows;

while (getline(stroka, text)) {

SinglyLinkedList<bool> shouldRemove;

for (int i = 0; i < stlbindex.size; ++i) {

stringstream iss(text);

string token;

int currentIndex = 0;

bool check = false;

while (getline(iss, token, ',')) {

if (currentIndex == stlbindex.getvalue(i) && token == conditions.getvalue(i).value) {

check = true;

break;

}

currentIndex++;

}

if (check) shouldRemove.push\_back(true);

else shouldRemove.push\_back(false);

}

if (conditions.getvalue(1).logicalOP == "and") { // Если оператор И

if (shouldRemove.getvalue(0) && shouldRemove.getvalue(1));

else filteredRows += text + "\n";

} else { // Если оператор ИЛИ

if (!(shouldRemove.getvalue(0)) && !(shouldRemove.getvalue(1))) filteredRows += text + "\n";

}

}

foutput(filepath, filteredRows);

copy--;

}

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "open");

cout << "Команда выполнена!" << endl;

} else cout << "Ошибка, таблица используется другим пользователем!" << endl;

}

// ф-ии инсерта

void isValidInsert(string& command) { // ф-ия проверки ввода команды insert

string table;

int position = command.find\_first\_of(' ');

if (position != -1) { // проверка синтаксиса

table = command.substr(0, position);

command.erase(0, position + 1);

if (nametables.getindex(table) != -1) { // проверка таблицы

if (command.substr(0, 7) == "values ") { // проверка values

command.erase(0, 7);

position = command.find\_first\_of(' ');

if (position == -1) { // проверка синтаксиса ///////

if (command[0] == '(' && command[command.size()-1] == ')') { // проверка синтаксиса скобок и их удаление

command.erase(0, 1);

command.pop\_back();

position = command.find(' ');

while (position != -1) { // удаление пробелов

command.erase(position);

position = command.find(' ');

}

insert(table, command);

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нет такой таблицы!" << endl;

} else cout << "Ошибка, нарушен синатксис команды" << endl;

}

void insert(string& table, string& values) { // ф-ия вставки в таблицу

string filepath;

int index = nametables.getindex(table); // получаем индекс таблицы(aka key)

if (checkLockTable(table)) {

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "close");

// вставка значений в csv, не забывая про увеличение ключа

filepath = "../" + nameBD + "/" + table + "/1.csv";

int countline = CountLine(filepath);

int fileid = 1; // номер файла csv

while (true) {

if (countline == tupleslimit) { // если достигнут лимит, то создаем/открываем другой файл

fileid++;

filepath = "../" + nameBD + "/" + table + "/" + to\_string(fileid) + ".csv";

if (fileindex.getvalue(index) < fileid) {

fileindex.replace(index, fileid);

}

} else break;

countline = CountLine(filepath);

}

fstream file;

file.open(filepath, ios::app);

int val = countlines.getvalue(index);

file << to\_string(val) + ',' + values + '\n';

val++;

countlines.replace(index, val);

file.close();

filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

foutput(filepath, "open");

cout << "Команда выполнена!" << endl;

} else cout << "Ошибка, таблица используется другим пользователем!" << endl;

}

// ф-ии селекта

void isValidSelect(string& command) { // ф-ия проверки ввода команды select

Where conditions;

SinglyLinkedList<Where> cond;

if (command.find\_first\_of("from") != -1) {

// работа со столбцами

while (command.substr(0, 4) != "from") {

string token = command.substr(0, command.find\_first\_of(' '));

if (token.find\_first\_of(',') != -1) token.pop\_back(); // удаляем запятую

command.erase(0, command.find\_first\_of(' ') + 1);

if (token.find\_first\_of('.') != -1) token.replace(token.find\_first\_of('.'), 1, " ");

else {

cout << "Ошибка, нарушен синтаксис команды!" << endl;

return;

}

stringstream ss(token);

ss >> conditions.table >> conditions.column;

bool check = false;

int i;

for (i = 0; i < nametables.size; ++i) { // проверка, сущ. ли такая таблица

if (conditions.table == nametables.getvalue(i)) {

check = true;

break;

}

}

if (!check) {

cout << "Нет такой таблицы!" << endl;

return;

}

check = false;

stringstream iss(stlb.getvalue(i));

while (getline(iss, token, ',')) { // проверка, сущ. ли такой столбец

if (token == conditions.column) {

check = true;

break;

}

}

if (!check) {

cout << "Нет такого столбца" << endl;

return;

}

cond.push\_back(conditions);

}

command.erase(0, command.find\_first\_of(' ') + 1); // скип from

// работа с таблицами

int iter = 0;

while (!command.empty()) { // пока строка не пуста

string token = command.substr(0, command.find\_first\_of(' '));

if (token.find\_first\_of(',') != -1) {

token.pop\_back();

}

int position = command.find\_first\_of(' ');

if (position != -1) command.erase(0, position + 1);

else command.erase(0);

if (iter + 1 > cond.size || token != cond.getvalue(iter).table) {

cout << "Ошибка, указаные таблицы не совпадают или их больше!" << endl;

return;

}

if (command.substr(0, 5) == "where") break; // также заканчиваем цикл если встретился WHERE

iter++;

}

if (command.empty()) {

select(cond);

} else {

if (command.find\_first\_of(' ') != -1) {

command.erase(0, 6);

int position = command.find\_first\_of(' ');

if (position != -1) {

string token = command.substr(0, position);

command.erase(0, position + 1);

if (token.find\_first\_of('.') != -1) {

token.replace(token.find\_first\_of('.'), 1, " ");

stringstream ss(token);

string table, column;

ss >> table >> column;

if (table == cond.getvalue(0).table) { // проверка таблицы в where

position = command.find\_first\_of(' ');

if ((position != -1) && (command[0] == '=')) {

command.erase(0, position + 1);

position = command.find\_first\_of(' ');

if (position == -1) { // если нет лог. операторов

if (command.find\_first\_of('.') == -1) { // если просто значение

conditions.value = command;

conditions.check = true;

selectWithValue(cond, table, column, conditions);

} else { // если столбец

command.replace(command.find\_first\_of('.'), 1, " ");

stringstream iss(command);

iss >> conditions.table >> conditions.column;

conditions.check = false;

selectWithValue(cond, table, column, conditions);

}

} else { // если есть лог. операторы

SinglyLinkedList<Where> values;

token = command.substr(0, position);

command.erase(0, position + 1);

if (token.find\_first\_of('.') == -1) { // если просто значение

conditions.value = token;

conditions.check = true;

values.push\_back(conditions);

} else { // если столбец

token.replace(token.find\_first\_of('.'), 1, " ");

stringstream stream(token);

stream >> conditions.table >> conditions.column;

conditions.check = false;

values.push\_back(conditions);

}

position = command.find\_first\_of(' ');

if ((position != -1) && (command.substr(0, 2) == "or" || command.substr(0, 3) == "and")) {

conditions.logicalOP = command.substr(0, position);

command.erase(0, position + 1);

position = command.find\_first\_of(' ');

if (position != -1) {

token = command.substr(0, position);

command.erase(0, position + 1);

if (token.find\_first\_of('.') != -1) {

token.replace(token.find\_first\_of('.'), 1, " ");

stringstream istream(token);

SinglyLinkedList<string> tables;

SinglyLinkedList<string> columns;

tables.push\_back(table);

columns.push\_back(column);

istream >> table >> column;

tables.push\_back(table);

columns.push\_back(column);

if (table == cond.getvalue(0).table) { // проверка таблицы в where

position = command.find\_first\_of(' ');

if ((position != -1) && (command[0] == '=')) {

command.erase(0, position + 1);

position = command.find\_first\_of(' ');

if (position == -1) { // если нет лог. операторов

if (command.find\_first\_of('.') == -1) { // если просто значение

conditions.value = command.substr(0, position);

conditions.check = true;

command.erase(0, position + 1);

values.push\_back(conditions);

selectWithLogic(cond, tables, columns, values);

} else { // если столбец

token = command.substr(0, position);

token.replace(token.find\_first\_of('.'), 1, " ");

command.erase(0, position + 1);

stringstream stream(token);

stream >> conditions.table >> conditions.column;

conditions.check = false;

values.push\_back(conditions);

selectWithLogic(cond, tables, columns, values);

}

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, таблица в where не совпадает с начальной!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

}

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, таблица в where не совпадает с начальной!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

}

} else cout << "Ошибка, нарушен синтаксис команды!" << endl;

}

void select(SinglyLinkedList<Where>& conditions) { // ф-ия обычного селекта

for (int i = 0; i < conditions.size; ++i) {

bool check = checkLockTable(conditions.getvalue(i).table);

if (!check) {

cout << "Ошибка, таблица открыта другим пользователем!" << endl;

return;

}

}

string filepath;

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "close");

}

SinglyLinkedList<int> stlbindex = findIndexStlb(conditions); // узнаем индексы столбцов после "select"

SinglyLinkedList<string> tables = textInFile(conditions); // записываем данные из файла в переменные для дальнейшей работы

sample(stlbindex, tables); // выборка

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "open");

}

}

void selectWithValue(SinglyLinkedList<Where>& conditions, string& table, string& stolbec, struct Where value) { // ф-ия селекта с where для обычного условия

for (int i = 0; i < conditions.size; ++i) {

bool check = checkLockTable(conditions.getvalue(i).table);

if (!check) {

cout << "Ошибка, таблица открыта другим пользователем!" << endl;

return;

}

}

string filepath;

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "close");

}

SinglyLinkedList<int> stlbindex = findIndexStlb(conditions); // узнаем индексы столбцов

int stlbindexval = findIndexStlbCond(table, stolbec); // узнаем индекс столбца условия

int stlbindexvalnext = findIndexStlbCond(value.table, value.column); // узнаем индекс столбца условия после '='(нужно если условиестолбец)

SinglyLinkedList<string> tables = textInFile(conditions); // записываем данные из файла в переменные для дальнейшей работы

SinglyLinkedList<string> column = findStlbTable(conditions, tables, stlbindexvalnext, value.table);; // записываем колонки таблицы условия после '='(нужно если условиестолбец)

// фильтруем нужные строки

for (int i = 0; i < conditions.size; ++i) {

if (conditions.getvalue(i).table == table) {

stringstream stream(tables.getvalue(i));

string str;

string filetext;

int iterator = 0; // нужно для условиястолбец

while (getline(stream, str)) {

stringstream istream(str);

string token;

int currentIndex = 0;

while (getline(istream, token, ',')) {

if (value.check) { // для простого условия

if (currentIndex == stlbindexval && token == value.value) {

filetext += str + '\n';

break;

}

currentIndex++;

} else { // для условиястолбец

if (currentIndex == stlbindexval && token == column.getvalue(iterator)) {

filetext += str + '\n';

}

currentIndex++;

}

}

iterator++;

}

tables.replace(i, filetext);

}

}

sample(stlbindex, tables); // выборка

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "open");

}

}

void selectWithLogic(SinglyLinkedList<Where>& conditions, SinglyLinkedList<string>& table, SinglyLinkedList<string>& stolbec, SinglyLinkedList<Where>& value) {

for (int i = 0; i < conditions.size; ++i) {

bool check = checkLockTable(conditions.getvalue(i).table);

if (!check) {

cout << "Ошибка, таблица открыта другим пользователем!" << endl;

return;

}

}

string filepath;

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "close");

}

SinglyLinkedList<int> stlbindex = findIndexStlb(conditions); // узнаем индексы столбцов после "select"

SinglyLinkedList<string> tables = textInFile(conditions); // записываем данные из файла в переменные для дальнейшей работы

SinglyLinkedList<int> stlbindexval;// узнаем индексы столбца условия

for (int i = 0; i < stolbec.size; ++i) {

int index = findIndexStlbCond(table.getvalue(i), stolbec.getvalue(i));

stlbindexval.push\_back(index);

}

SinglyLinkedList<int> stlbindexvalnext; // узнаем индекс столбца условия после '='(нужно если условиестолбец)

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

int index = findIndexStlbCond(value.getvalue(i).table, value.getvalue(i).column); // узнаем индекс столбца условия после '='(нужно если условиестолбец)

stlbindexvalnext.push\_back(index);

}

SinglyLinkedList<string> column;

for (int j = 0; j < value.size; ++j) {

if (!value.getvalue(j).check) { // если условие столбец

column = findStlbTable(conditions, tables, stlbindexvalnext.getvalue(j), value.getvalue(j).table);

}

}

// фильтруем нужные строки

for (int i = 0; i < conditions.size; ++i) {

if (conditions.getvalue(i).table == table.getvalue(0)) {

stringstream stream(tables.getvalue(i));

string str;

string filetext;

int iterator = 0; // нужно для условиястолбец

while (getline(stream, str)) {

SinglyLinkedList<bool> checkstr;

for (int j = 0; j < value.size; ++j) {

stringstream istream(str);

string token;

int currentIndex = 0;

bool check = false;

while (getline(istream, token, ',')) {

if (value.getvalue(j).check) { // если просто условие

if (currentIndex == stlbindexval.getvalue(j) && token == value.getvalue(j).value) {

check = true;

break;

}

currentIndex++;

} else { // если условие столбец

if (currentIndex == stlbindexval.getvalue(j) && token == column.getvalue(iterator)) {

check = true;

break;

}

currentIndex++;

}

}

checkstr.push\_back(check);

}

if (value.getvalue(1).logicalOP == "and") { // Если оператор И

if (checkstr.getvalue(0) && checkstr.getvalue(1)) filetext += str + "\n";

} else { // Если оператор ИЛИ

if (!checkstr.getvalue(0) && !checkstr.getvalue(1));

else filetext += str + "\n";

}

iterator++;

}

tables.replace(i, filetext);

}

}

sample(stlbindex, tables); // выборка

for (int i = 0; i < conditions.size; ++i) {

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + conditions.getvalue(i).table + "\_lock.txt";

foutput(filepath, "open");

}

}

// Вспомогательные ф-ии, чтобы избежать повтора кода в основных ф-иях

bool checkLockTable(string table) { // ф-ия проверки, закрыта ли таблица

string filepath = "../" + nameBD + "/" + table + "/" + table + "\_lock.txt";

string check = finput(filepath);

if (check == "open") return true;

else return false;

}

SinglyLinkedList<int> findIndexStlb(SinglyLinkedList<Where>& conditions) { // ф-ия нахождения индекса столбцов(для select)

SinglyLinkedList<int> stlbindex;

for (int i = 0; i < conditions.size; ++i) {

int index = nametables.getindex(conditions.getvalue(i).table);

string str = stlb.getvalue(index);

stringstream ss(str);

int stolbecindex = 0;

while (getline(ss, str, ',')) {

if (str == conditions.getvalue(i).column) {

stlbindex.push\_back(stolbecindex);

break;

}

stolbecindex++;

}

}

return stlbindex;

}

int findIndexStlbCond(string table, string stolbec) { // ф-ия нахождения индекса столбца условия(для select)

int index = nametables.getindex(table);

string str = stlb.getvalue(index);

stringstream ss(str);

int stlbindex = 0;

while (getline(ss, str, ',')) {

if (str == stolbec) break;

stlbindex++;

}

return stlbindex;

}

SinglyLinkedList<string> textInFile(SinglyLinkedList<Where>& conditions) { // ф-ия инпута текста из таблиц(для select)

string filepath;

SinglyLinkedList<string> tables;

for (int i = 0; i < conditions.size; ++i) {

string filetext;

int index = nametables.getindex(conditions.getvalue(i).table);

int iter = 0;

do {

iter++;

filepath = "../" + nameBD + '/' + conditions.getvalue(i).table + '/' + to\_string(iter) + ".csv";

string text = finput(filepath);

int position = text.find('\n'); // удаляем названия столбцов

text.erase(0, position + 1);

filetext += text + '\n';

} while (iter != fileindex.getvalue(index));

tables.push\_back(filetext);

}

return tables;

}

SinglyLinkedList<string> findStlbTable(SinglyLinkedList<Where>& conditions, SinglyLinkedList<string>& tables, int stlbindexvalnext, string table) { // ф-ия инпута нужных колонок из таблиц для условиястолбец(для select)

SinglyLinkedList<string> column;

for (int i = 0; i < conditions.size; ++i) {

if (conditions.getvalue(i).table == table) {

stringstream stream(tables.getvalue(i));

string str;

while (getline(stream, str)) {

stringstream istream(str);

string token;

int currentIndex = 0;

while (getline(istream, token, ',')) {

if (currentIndex == stlbindexvalnext) {

column.push\_back(token);

break;

}

currentIndex++;

}

}

}

}

return column;

}

void sample(SinglyLinkedList<int>& stlbindex, SinglyLinkedList<string>& tables) { // ф-ия выборки(для select)

for (int i = 0; i < tables.size - 1; ++i) {

stringstream onefile(tables.getvalue(i));

string token;

while (getline(onefile, token)) {

string needstlb;

stringstream ionefile(token);

int currentIndex = 0;

while (getline(ionefile, token, ',')) {

if (currentIndex == stlbindex.getvalue(i)) {

needstlb = token;

break;

}

currentIndex++;

}

stringstream twofile(tables.getvalue(i + 1));

while (getline(twofile, token)) {

stringstream itwofile(token);

currentIndex = 0;

while (getline(itwofile, token, ',')) {

if (currentIndex == stlbindex.getvalue(i + 1)) {

cout << needstlb << ' ' << token << endl;

break;

}

currentIndex++;

}

}

}

}

}

};

int main() {

DataBase carshop;

carshop.parse();

carshop.mkdir();

string command;

while (true) {

cout << endl << "Введите команду: ";

getline(cin, command);

carshop.checkcommand(command);

}

return 0;

}

# Вывод

В ходе выполнения практической работы №1 я познакомился с понятием СУБД, SQL базы данных. Научился реализовывать свою собственную реляционную СУБД при поддержки некоторых запросов на языке SQL.