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*

*Деревянкин Д.А. Архипова А. Б.*

«17» января 2025 г.«\_\_\_» \_\_\_\_\_\_ 2025 г.

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

(подпись) (подпись)

Новосибирск 2025

**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:

{

"name": "СетьМагазинов",

"tuples\_limit": 5,

"structure": {

"shops": ["Город", "Район", "Улица", "Дом"],

"users": ["Фамилия", "Имя", "Номер Заказа"]

}

}

## headers:

includes.h

#ifndef utility\_h

#define utility\_h

using namespace std;

#include <iostream>

#include <fstream>

#include <sstream>

#include <filesystem>

#include <functional>

#endif // UTILITY\_H

list.h

#ifndef list\_h

#define list\_h

#include "includes.h"

template <typename T>

struct Node {

T data;

Node\* next; //Указатель на след. элемент

Node(T value); //Конструктор узла

};

template <typename T>

struct SinglyLinkedList {

Node<T>\* head;

size\_t elementCount = 0;

SinglyLinkedList(); //Конструктор

~SinglyLinkedList(); //Деконструктор

bool isEmpty() const;

void print(); // ф-ия вывода списка

void pushFront(T value); //Добавление в начало списка

void pushBack(T value); //Добавление в конец списка

void popFront(); //Удаление в начале списка

void popBack(); //Удаление в конце списка

void removeAt(T value); //Удаление по индексу

bool find(T value); //Поиск значений в списке

void clearSList();

T getElementAt(int index) const;

int getIndex(T value);

void replace(int index, T newValue);

Node<T>\* getHead() const;

size\_t size();

};

#include "../src/list.cpp"

#endif // LIST\_H

## 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; //Если список не содержит элементов, то присваиваем нулевое значение

}

template <typename T>

SinglyLinkedList<T>::~SinglyLinkedList() {

while(!isEmpty()) {

popFront(); //Вызывает список, после которого очищает память

}

}

template <typename T>

bool SinglyLinkedList<T>::isEmpty() const {

return elementCount == 0;

}

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>::pushFront(T value) {

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

newNode->next = head; //Следующий узел, становится текущей головой списка

head = newNode;

elementCount++;

}

template <typename T>

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

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

if(head == nullptr) { //Условие если список пуст

head = newNode;

} else {

Node<T>\* current = head;

while(current->next != nullptr) {

current = current->next; //Присваиваем current следующее значение

}

current->next = newNode;

}

elementCount++;

}

template <typename T>

void SinglyLinkedList<T>::popFront() {

if(head == nullptr) return;

Node<T>\* nextHead = head->next; //Создание временной переменной, для сохранения

delete head;

head = nextHead;

elementCount--;

}

template <typename T>

void SinglyLinkedList<T>::popBack() {

if (head == nullptr) return; // Если список пуст

if (head->next == nullptr) { // Если один элемент

delete head;

head = nullptr; // Исправлено

} else {

Node<T>\* current = head;

while (current->next->next != nullptr) {

current = current->next;

}

delete current->next; // Удаляем последний элемент

current->next = nullptr; // Обнуляем указатель на следующий элемент

}

elementCount--;

}

template <typename T>

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

if (isEmpty()) return;

if (head->data == value) {

popFront();

return;

}

Node<T>\* current = head;

while (current->next != nullptr && current->next->data != value) {

current = current->next;

}

if (current->next != nullptr) {

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

current->next = nodeToDelete->next;

delete nodeToDelete;

elementCount--;

}

}

template <typename T>

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

Node<T>\* current = head;

while (current != nullptr) {

if (current->data == value) {

return true;

}

current = current->next;

}

return false;

}

template <typename T>

void SinglyLinkedList<T>::clearSList() {

while (!isEmpty()) {

popFront();

}

}

template <typename T>

T SinglyLinkedList<T>::getElementAt(int index) const {

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

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>

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

Node<T>\* current = head;

int index = 0;

while (current) {

if (current->data == value) {

return index; // Возвращаем индекс, если значение найдено

}

current = current->next;

index++;

}

return -1; // Возвращаем -1, если значение не найдено

}

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>

Node<T>\* SinglyLinkedList<T>::getHead() const {

return head;

}

template <typename T>

size\_t SinglyLinkedList<T>::size() {

return elementCount;

}

# 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.

Ссылка на репозиторий: [нажмите](https://github.com/WarpMiner/Pract1/)