**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра РАПС**

**Курсовая работа**

**по дисциплине «Программирование и основы алгоритмизации»**

**Тема: Реализация интерфейсного приложения с БД на языке С++**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9404 |  | Семыкин В. |
| Преподаватель |  | Армашев А. А. |

Санкт-Петербург

2020

**Содержание**

Введение…………………………………………………………………………………………………3

1. Описание разработки.………………………………………………………………………………..4

1.1 Сбор информации…………………………………………………………………………...4

1.2 Парсинг .html документа, взятого с сайта ранее указанным методом…………………..4

1.3 Пользовательский интерфейс программы………………………………………………...5

1.4 Функции для работы с базой данных……………………………………………………...5

2. Описание функция для работы с базой данных……………………………………………………6

2.1 Создание базы данных……………………………………………………………………...6

2.2 Вывод данных в табличном виде в окне…………………………………………………..6

2.3 Добавление записи в конец базы данных…………………………………………………6

2.4 Добавление записи в определенную позицию базы данных……………………………7

2.5 Изменение определенной записи…………………………………………………………..7

2.6 Удаление определенной записи…………………………………………………………….7

2.7 Удаление всех записей……………………………………………………………………...7

2.8 Шифрование данных………………………………………………………………………..8

2.9 Дешифрование данных……………………………………………………………………..8

2.10 Поиск записей по входным данным……………………………………………………...8

2.11 Сортировка записей……………………………………………………………………….9

2.12 Сохранение состояния базы данных……………………………………………………..9

2.13 Резервное копирование базы данных…………………………………………………….9

2.14 Обновление окна представления данных………………………………………………...9

2.15 Выход из программы………………………………………………………………………9

3. Модульная структура программы………………………………………………………………….11

4. Заключение………………………………………………………………………………………….12

5. Вывод………………………………………………………………………………………………..13

6. Список литературы…………………………………………………………………………………14

7. Приложения…………………………………………………………………………………………15

**Введение**

Цель работы заключается в получении навыков постановки задачи, алгоритмизации, модульного принципа разработки и отладки приложений на примере создания программы, имеющей пользовательский интерфейс, для работы с базой данных.

В качестве баз данных я создаю 2 файла никак не связанных между собой. В которых хранятся по одной таблице с данными. Файлы имеют расширение **.sql**. Таблица в первом файле содержит в себе данные о пользователях (ID, логин и пароль). Таблица во втором файле содержит в себе данные о погоде (ID, день недели, дата, месяц, основные характеристики и температура). Я создал два файла для наглядности, но, конечно же можно было все сделать в одном файле, просто не задавать между таблицами связь.

Основными задачами являются создание программы, которая работает с базой данных в виде sql файлов. Код программы должен поддерживать модульный принцип разработки. При реализации программы необходимо использовать ООП.

СУБД (система управления базой данных) должна выполнять следующие действия:

1. просмотр базы данных;
2. поиск записей по входным параметрам;
3. редактирование записи, по разным входным параметрам;
4. добавление записи в конец списка и в любое место таблицы;
5. удаление записи, номер которой задан пользователем;
6. удаление всех записей;
7. сортировку записей;
8. шифрование и дешифрование данных;
9. сохранение состояния базы данных;

10) обновление окна отображения даных;

Также СУБД обязательно должна предусамотривать обработку ошибочных ситуаций.

В свой работе я использовал тему – “Анализ метеоданных”.

**Описание разработки**

**1.1 Сбор информации**

Сбор информации в моей программе производился при помощи специальной библиотеки libcurl (https://curl.haxx.se/libcurl/c/) с сайта [https://sinoptik.ua](https://sinoptik.ua/). Погода взята с города Санкт-Петербург. Работа с библиотекой libcurl осуществлялась в файле htmlThief.h, протестировать который можно в заранее скомпиллированном htmlThief.sh файле. Принцип работы заключается в следующем:

1) Объявление флага CURL\_GLOBAL\_ALL, который отвечает за подключение всевозможных протоколов передачи данных в интернете, таких как UDP, TCP/IP, устаревший TLS, современный SSL, HTTP, HTTPS, FTP и других;

2) Инициализация сессии curl’а, которая подразумевает под собой возвращение стандартного дескриптора CURL, который в свою очередь понадобится для входного сигнала для других функций;

3) Далее идет установка опций:

a) Установка URL адреса нужной страницы, в моем случае - <https://sinoptik.com.ru/погода-санкт-петербург>;

b) Включение полного вывода протокола/отладки во время работы;

c) Включение прогресс-бара (не обязательная опция);

d) Включение опции, отвечающей за запись информации в заранее подготовленной статической функции writeData типа size\_t, которая возвращает кол-во записанных байт в файловый поток FILE\*, поэтому writeData является типом size\_t (целочисленное).

4) Открытие файла в режиме (“wb”), для записи в него данных в бинарном формате. Далее функциями curl’а записываем в файл расширения .html данные, “сворованные” с сайта, указанного раннее, то есть небольшой web scrapping;

5) Вызов функции curl\_easy\_cleanup(CURL\* handle) для “очитки” потока, который мы объявляли в самом начале работы с данной библиотекой. Тот же принцип, что и закрытие файла;

6) Далее идет функция схожая с предыдущей. Она очищает глобальный объявленный поток CURL (применяется только если была объявлена функция, которая инициализирует глобальный поток, также не является безопасной, если вы не объявляли глобальный поток curl\_global\_init(void)).

**1.2 Парсинг .html документа, взятого с сайта ранее указанным методом**

Для парсинга была использована библиотека, написанная на чистом С, компанией Microsoft – Gumbo-Parser (http://matze.github.io/clib-doc/gumbo-parser/index.html). Парсинг .html документа заключается в нахождении в нём нужных характеристик для будущей базы данных,затем добавление данных в массив структур. Принцип парсинга заключается в следующем:

a) Каждая функция, производимая поиск нужных данных (таких функций несколько и принцип работы у них практически идентичный, отличается лишь тем, что при изучении раннее взятого .html документа было обнаружено, что нужные данные были заключены в разных тегах, у некоторых тегов были классы, естественно, производился поиск по названию класса тега.) включала в себя проверку тега. Тег был идентифицирован при изучении .html документа самостоятельно;

b) Проверка на тег (является ли приходимый параметр {узел} в функцию нужным тегом), также проверка на имя класса (что не является для всех функций обязательным, ибо не у всех есть класс);

c) Объявление еще одного узла GumboNode\* и присвоение ему данных дочернего элемента;

d) Проверка типа узла (надо убедиться, что узел является GUMBO\_NODE\_TEXT, то есть текстовой информацией, также проверка на GUMBO\_NODE\_WHITESPACE, чтобы не было пустого текста);

e) Копирование текста в массив структур;

f) Проверка в цикле (есть ли еще дочерние элементы, затем вызов той функции, в которой шла работа, то есть рекурсия).

**1.3 Пользовательский интерфейс программы**

Во время запуска программы появляется окно входа в систему, также можно в этом же окне зарегистрироваться. При попытке входа программа сравнивает записи в базе данных с пользователями и сопосталяет их с введенными данными для входа, если проверка прошла успешно, выполняется вход, если нет, то программа выдает окно с соответсвующим сообщеним. Во время регистрации вы вводите логин и два раза пароль. Программа добавляет вас в базу данных и уведомляет об этом. Также, обработана следующая ошибка: когда вы вводите пароли, проверяется их соответствие. В программе заранее создан привилегилированный пользователь – админ, при входе за которого можно увидеть специальную панель управления админа. В ней, соответственно, можно производить самые различные изменения в базе данных.

**1.4 Функции для работы с базой данных**

В работу включены такие функции как:

- Создание базы данных;

- Вывод данных в табличном виде в окне;

- Добавление записи в конец базы данных;

- Добавлениe записи в определенную позицию базы данных;

- Изменение определенной записи;

- Удаление определенной записи;

- Удаление всех записей;

- Зашифровать файл;

- Дешифровать файл;

- Поиск записей по входным данным;

- Сортировка записей;

- Сохранение состояния базы данных;

- Резервное копирование базы данных;

- Обновление окна представления данных;

- Выход из программы.

**2.1 Создание базы данных**

Создание базы данных происходит по довольно простому алгоритму. Создается объект класса **QsqlDataBase**, далее просходит добавление базы данных, используя **QSQLITE**. Устанавливаем имя хоста для базы данных и просто ее имя самой базы данных, которое является непосредственно путем к базе данных.

**2.2 Вывод данных в табличном виде в окне**

Вывод данных происходит в окне, для этого используется класс **tableView,** в котором можно настроить вид отображаемых данных. Например, в своей работе я произвел следующую настройку:

**setModel –** устанавливаю модель на мой элемент формы **tableView;**

**setSelectionBehavior –** режим выделения строк **(SelectRows);**

**setSelectionMode –** режим выделения лишь одной строки **(SingleSelection);**

**resizeColumnsToContents –** устанавливаю размер колонок по размеру содержимого;

**setEditTriggers –** режим не позволяющий изменять данные посредством двойного кликанья на определенную строку и колонку, и написания других данных **(NoEditTriggers);**

**setStretchLastSection –** режим, позволяющий заполнить доступное пространство, растянув последний раздел заголовка **(установил на true);**

**2.3 Добавление записи в конец базы данных**

Для добавления записи в конец базы данных не требуется особых навыков написания sql-запросов. То есть, для добавления записи в базу данных (любую), я написал простейший sql-запрос, в следующем виде:

INSERT INTO Weather (Weekday, Date\_, Month, Label, Temperature)

VALUES (…)

Данные для заполнения таблицы приходят в метод параметром, созданным от класса **QVariantList.**

**2.4 Добавление записи в определенную позицию базы данных**

Для добавления записи в определенную позицию, просто добавляем в предыдущий запрос нужный ID, то есть нужный номер строки в которую необходимо добавить запись:

INSERT INTO Weather (ID, Weekday, Date\_, Month, Label, Temperature)

VALUES (…)

Данные для заполнения таблицы приходят в метод параметром, созданным от класса **QVariantList.**

**2.5 Изменение определенной записи**

Для изменения используется также sql-запрос, включающий в себя sql оператор UPDATE. Пример запроса для обновления записи (естественно метод работает с входящим параметром ID, чтобы изменить какую-то определенную запись):

UPDATE Weather

SET Weekday = …,

Date\_ = …,

Month = …,

Label = …,

Temperature = …

WHERE ID = …

Под многоточием подразумевается строки данных, которые заменят старые данные.

**2.6 Удаление определенной записи**

Для удаления используется оператор DELETE. Пример удаления определенной записи в базе данных:

DELETE

FROM Weather

WHERE ID = …

Опять же я использую ID, как надежный ключ к строке, которую необходимо удалить.

**2.7 Удаление всех записей**

Для удаления всех записей используется более интересный и хитрый составной запрос, включающий в себя сразу 3 запроса. Пример такого запроса можно увидеть ниже у меня в классе **DataBase**, в методе **deleteAllWeatherRecords():**

**1 запрос:**

DELETE FROM Weather

Данный запрос просто удаляет все записи, причем не удаляя ID, которые были заняты.

**2 запрос:**

VACUUM

Следующий запрос перестраивает файл базы данных, “переупаковывая” его в минимальный объем дискового пространства. Запуск VACUUM для перестройки базы данных освобождает пространство и уменьшает размер файла базы данных. Частые вставки, обновления и удаления могут привести к фрагментации файла базы данных, когда данные для одной таблицы или индекса разбросаны по всему файлу базы данных. Запуск VACUUM гарантирует, что каждая таблица и индекс в значительной степени хранятся непрерывно в файле базы данных. В некоторых случаях VACUUM может также уменьшить количество частично заполненных страниц в базе данных, еще больше уменьшая размер файла базы данных.

**3 запрос:**

DROP TABLE Weather

Данная команда просто удаляет таблицу, это сделано для ранее описанной проблемы с занятыми ID.

Далее, необходимо после метода удаления использовать метод создания таблицы.

**2.8 Шифрование данных**

В методе шифрования я использовал широко известное шифрование XOR, то есть шифрование бинарным оператором ИСКЛЮЧАЮЩЕЕ ИЛИ. В моей работе код шифрования можно найти в классе **MainDataBase,** в методе **xor\_encryption(const std::string &key, std::string &str);** Работает шифрование таким образом:

str[i] ^= key[i % key.size()];

То есть, мы используем бинарный оператор исключающее или к каждому символу строки на ключ с индексом, идентичным индексу символа строки, от которого берем остаток деления индекса на размер ключа. Ключ тоже является строкой, как можно заметить выше по объявлению метода.

**2.9 Дешифрование данных**

Дешифрование данных происходит повторным шифрованием строк. В своей работе я встретился с проблемой: во время шифрования строки шифрующими символами являлись символы переноса строки (пытался заменять этот символ сепараторами, но это не помогло), из-за чего повторное шифрование выдавало неполную картину. Данные были странно расшифрованы. Исправил я это следующим образом: я сохраняю полностью строку со всеми данными и ее два раза шифрую, то есть дешифрую, чтобы наглядно продемонстрировать, что метод действительно работает. Затем перезаписываю данные в таблицу.

**2.10 Поиск записей по входным данным**

Методы поиска реализовал через прочтение данных из файла введенных пользователем как входной параметр. То есть пользователь ввел. Например, день недели – Понедельник. Я произвожу чтение из файла sql-запросом, в котором оператором WHERE задаю этот самый Понедельник. Далее срабатывает запрос. Удаляю все данные из табличного представления данных и заполняю новыми, взятыми из ранее описанного запроса.

**2.11 Сортировка записей**

Алгортитм сортировки довольно прост в моей работе. Я использовал sql-запрос с оператором ORDER BY, который производит сортировку по столбцам таблицы. Пример сортировки:

SELECT \*

FROM Weather

ORDER BY Month

Такой запрос вернет табличное представление всех данных, отсортированных по месяцам.

**2.12 Сохранение состояния базы данных**

Алгортитм сохранения работает следующим образом: перед произведениями любых изменений в базе данных происходит резервное копирование всех данных, которое будет описанно далее в пункте 2.12. То есть перед каждым изменением идет сохранения начального состояния, чтобы ничего данные не потеряли структурированность и не произошла утечка памяти. Все данные сохраняются в переменную класса **QVariantList** для удобного дальнейшего использования данных.

**2.13 Резервное копирование базы данных**

Резервное копирование представляет собой полное считывание данных из таблицы и сохранения их в переменную класса **QVariantList.** Для подробного рассмотрения данного метода, рекомендуется просмотреть файл **database.cpp,** класс **DataBase,** метод **backupDataFromWeatherDB().**

**2.14 Обновление окна представления данных**

Алгоритм обновления окна представления данных довольно примитивен. Необходимо закрыть окно, затем открыть, для того, чтобы подгрузились данные и произведенные изменения.

Пример обновления:

this→close();

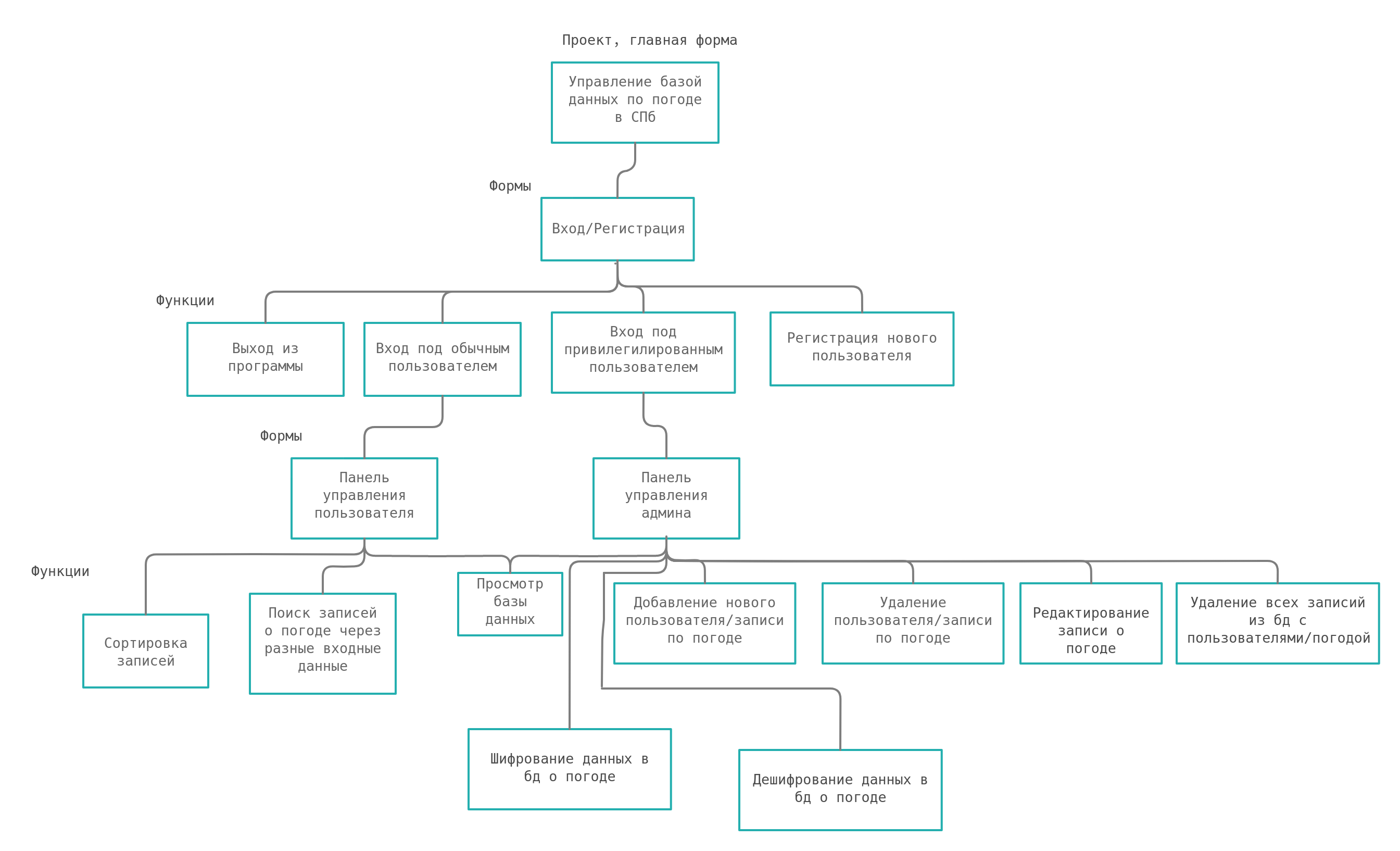
this→show();

**2.15 Выход из программы**

Алгоритм выхода из программы еще более прост. Требуется всего лишь выйти из приложения. Это делается одной строчкой:

QApplication::exit(EXIT\_SUCCESS);

EXIT\_SUCCESS это константа, объявленная командой define, эквивалентная по значению нулю, то есть успешный выход из программы, в файле **stdlib.h** или же в более новой версии, С++ файле **cstdlib.**

**Модульная структура программы**

**Рис. 1 –** “Модульная структура программы”

**Заключение**

В ходе написания курсовой работы был получен навык постановки задачи, алгоритмизации, модульного принципа разработки и отладки приложения на примере создания программы с пользовательским интерфейсом для работы с базой данных в виде sql файла (sql базы данных).

**Вывод**

Реализована простейшая программа на Qt/С++ для управления базы данных в виде программы с пользовательским интерфейсом. Была использована такая IDE (интегрированная среда разработки), как Qt. В ходе выполнения курсовой работы было придумано множество способов для манипулирования базой данных. Некоторые преимущества и недостатки тех или иных функций были описаны ранее в разделе – “Описание”. Было предусмотрено множество исключительных ситаций, чтобы избежать некорректную работоспособность программы. Также, код был залит на платформу Github – https://github.com/ViNN280801/CourseWork2.git

**Список литературы**

1.1 - Определенного автора нет. Libcurl man page. Curl license. 1997 г.

1.2 – Определенного автора нет. Gumbo-Parser. Microsoft copyright. 2005 г.

1.3 – Макс Шлее. Qt 5.10 Профессиональное программирование на C++. Санкт-Петербург “БХВ-Петербург” 2018 г.

1.4 – Авторы – CoderLessons. <https://coderlessons.com/tutorials/bazy-dannykh/vyuchit-sqlite/uchebnik-po-sqlite>

2019 г.

**Приложения**

**1.1 Код в файле addingnewweatherrecord.h:**

#ifndef ADDINGNEWWEATHERRECORD\_H

#define ADDINGNEWWEATHERRECORD\_H

#include <QWidget>

#include <QString>

#include <QMessageBox>

#include <QVariantList>

#include "database.h"

namespace Ui {

class AddingNewWeatherRecord;

}

class AddingNewWeatherRecord : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit AddingNewWeatherRecord(QWidget \*parent = nullptr);

~*AddingNewWeatherRecord*();

private slots:

void on\_addRecordPushButton\_clicked();

private:

Ui::AddingNewWeatherRecord \*ui;

DataBase \*db;

};

#endif // ADDINGNEWWEATHERRECORD\_H

**1.2 Код в файле addingnewweatherrecord.cpp:**

#include "addingnewweatherrecord.h"

#include "ui\_addingnewweatherrecord.h"

AddingNewWeatherRecord::AddingNewWeatherRecord(QWidget \*parent) :

QWidget(parent),

ui(new Ui::AddingNewWeatherRecord)

{

ui->setupUi(this);

db = new DataBase();

}

AddingNewWeatherRecord::~*AddingNewWeatherRecord*()

{

delete ui;

}

void AddingNewWeatherRecord::on\_addRecordPushButton\_clicked()

{

QVariantList data;

QString weekday = ui->weekdayLineEdit->text();

QString date = ui->dateLineEdit->text();

QString month = ui->monthLineEdit->text();

QString label = ui->labelLineEdit->text();

QString temperature = ui->temperatureLineEdit->text() + "°";

QString whichLine = ui->whichLineLineEdit->text();

for(int i = 0; i < weekday.length(); i++){

if(weekday[i] == ' ')

weekday[i] = '\_';

}

for(int i = 0; i < date.length(); i++){

if(date[i] == ' ')

date[i] = '\_';

}

for(int i = 0; i < month.length(); i++){

if(month[i] == ' ')

month[i] = '\_';

}

for(int i = 0; i < label.length(); i++){

if(label[i] == ' ')

label[i] = '\_';

}

for(int i = 0; i < temperature.length(); i++){

if(temperature[i] == ' ')

temperature[i] = '\_';

}

data.append(weekday);

data.append(date);

data.append(month);

data.append(label);

data.append(temperature);

if(db->checkingForIDInWeatherDB(ui->whichLineLineEdit->text().toInt())){

if(whichLine == nullptr){

if(weekday != nullptr && date != nullptr && month != nullptr &&

label != nullptr && temperature != nullptr){

db->insertIntoWeatherTable(data);

QMessageBox msgBox;

msgBox.setWindowTitle("Добавление");

msgBox.setText("Новая запись добавлена успешно.");

msgBox.*exec*();

this->close();

}

else if(weekday == nullptr && date == nullptr && month == nullptr &&

label == nullptr && temperature == nullptr){

QMessageBox::warning(this, "Внимание", "Не все поля заполнены.");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else if(whichLine != nullptr){

db->insertIntoWeatherTableWithID(data, whichLine.toInt());

QMessageBox msgBox;

msgBox.setWindowTitle("Добавление");

msgBox.setText("Новая запись добавлена успешно.");

msgBox.*exec*();

if(weekday == nullptr && date == nullptr && month == nullptr &&

label == nullptr && temperature == nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Запись с введенным ID уже существует, невозможно заменить.");

msgBox.*exec*();

}

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else if(!db->checkingForIDInWeatherDB(ui->whichLineLineEdit->text().toInt())){

QMessageBox::warning(this, "Внимание", "Вы ввели неверный ID записи.");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

ui->weekdayLineEdit->clear();

ui->dateLineEdit->clear();

ui->monthLineEdit->clear();

ui->labelLineEdit->clear();

ui->temperatureLineEdit->clear();

ui->whichLineLineEdit->clear();

this->close();

data.clear();

}

**2.1 Код файла database.h:**

#ifndef DATABASE\_H

#define DATABASE\_H

#include <QObject>

#include <QSql>

#include <QSqlQuery>

#include <QSqlError>

#include <QSqlDatabase>

#include <QFile>

#include <QDate>

#include <QDebug>

#include <ctime>

#define DATABASE\_HOSTNAME "UsersDataBase"

#define DATABASE\_NAME "users.sql"

#define TABLE "Users"

#define TABLE\_LOGIN "Login"

#define TABLE\_PSSWD "Password"

#define DATABASE\_HOSTNAME\_WEATHER "WeatherDataBase"

#define DATABASE\_WEATHER\_NAME "weather.sql"

#define TABLE\_WEATHER "Weather"

#define TABLE\_WEEKDAY "Weekday"

#define TABLE\_DATE "Date\_"

#define TABLE\_MONTH "Month"

#define TABLE\_LABEL "Label"

#define TABLE\_TEMPERATURE "Temperature"

class DataBase : public QObject

{

Q\_OBJECT

public:

explicit DataBase(QObject \*parent = 0);

virtual ~*DataBase*();

bool createMainTable();

void connectToDataBase();

bool insertIntoMainTable(const QVariantList &data);

bool insertIntoMainTableWithID(const QVariantList &data);

bool insertIntoMainTableWithKnownID(const QVariantList &data, int ID);

bool checkingForRecord(QString login);

bool checkingForID(int ID);

bool comparisonMethodIDWithLogin(int ID, QString login);

void deleteUserByID(int ID);

void deleteUserByLogin(QString login);

void deleteUserByIDAndLogin(int ID, QString login);

QString checkingForRecordReturningPsswd(QString login);

int checkUsersDataBaseCountOfRecords();

bool checkingForIDInWeatherDB(int ID);

bool checkingForWeekdayInWeatherDB(QString weekday);

bool insertIntoWeatherTable(const QVariantList &data);

bool insertIntoWeatherTableWithID(const QVariantList &data, int ID);

bool insertIntoWeatherTableWithKnownID(const QVariantList &data, int ID);

bool openWeatherDataBase();

bool createWeatherTable();

bool restoreWeatherDataBase();

void connectToWeatherDataBase();

void deleteWeatherRecordByID(int ID);

void deleteAllWeatherRecords();

void modifyWeatherRecordWithID(const QVariantList &data, int ID);

int checkWeatherDataBaseCountOfRecords();

int checkWeatherDataBaseCountOfWeekdayRecords(QString weekday);

void backupDataFromWeatherDB();

void undoMethod();

void searchByID(int ID);

void searchByWeekday(QString weekday);

void searchByDate(QString date);

void searchByMonth(QString month);

void closeDataBase();

void closeWeatherDataBase();

void setSaved(QVariantList savedData);

QVariantList getSaved();

private:

QSqlDatabase db;

QSqlDatabase weather\_db;

private:

bool openDataBase();

bool restoreDataBase();

};

class DataBaseReadMethods : public DataBase{

Q\_OBJECT

public:

explicit DataBaseReadMethods(QObject \*parent = 0);

virtual ~*DataBaseReadMethods*();

QVariantList readDataFromWeatherSqlFileToQVariantList();

QVariantList readDataFromUsersSqlFileToQVariantList();

QVariantList readWeekdayFromWeatherSqlFileToQVariantList(QString weekday);

QVariantList readDateFromWeatherSqlFileToQVariantList(QString date);

QVariantList readMonthFromWeatherSqlFileToQVariantList(QString month);

};

class SortMethods : public DataBaseReadMethods{

Q\_OBJECT

public:

explicit SortMethods(QObject \*parent = 0);

virtual ~*SortMethods*();

void sortingByWeekday();

void sortingByDate();

void sortingByMonth();

void sortingByGeneralCharacteristics();

void sortingByTemperature();

};

#endif // DATABASE\_H

**2.2 Код файла database.cpp:**

#include "database.h"

#include <QMessageBox>

QVariantList saved;

DataBase::DataBase(QObject \*parent) : QObject(parent)

{

}

DataBase::~*DataBase*(){

saved.clear();

}

DataBaseReadMethods::DataBaseReadMethods(QObject \*parent) : DataBase(parent)

{

}

DataBaseReadMethods::~*DataBaseReadMethods*()

{

}

SortMethods::SortMethods(QObject \*parent) : DataBaseReadMethods(parent)

{

}

SortMethods::~*SortMethods*()

{

}

void DataBase::connectToDataBase()

{

if(!QFile("/home/vladislav/Documents/StudyDocs/"

"3 semester/IT/Lab3/DataBaseOfUsers/DB/" DATABASE\_NAME).exists()){

this->restoreDataBase();

} else {

this->openDataBase();

}

}

bool DataBase::restoreDataBase()

{

this->closeWeatherDataBase();

if(this->openDataBase()){

if((!this->createMainTable())){

return false;

} else {

return true;

}

} else {

return false;

}

return false;

}

bool DataBase::openDataBase()

{

db = QSqlDatabase::addDatabase("QSQLITE");

db.setHostName(DATABASE\_HOSTNAME);

db.setDatabaseName("/home/vladislav/Documents/StudyDocs/"

"3 semester/IT/Lab3/DataBaseOfUsers/DB/" DATABASE\_NAME);

if(db.open()){

return true;

} else {

return false;

}

return false;

}

void DataBase::closeDataBase()

{

db.close();

}

void DataBase::closeWeatherDataBase()

{

weather\_db.close();

}

bool DataBase::createMainTable()

{

QSqlQuery query;

if(!query.exec("CREATE TABLE " TABLE " ("

"ID INTEGER PRIMARY KEY AUTOINCREMENT, "

TABLE\_LOGIN " TEXT NOT NULL,"

TABLE\_PSSWD " TEXT NOT NULL )")){

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoMainTable(const QVariantList &data)

{

this->connectToDataBase();

QSqlQuery query;

query.prepare("INSERT INTO " TABLE " ( " TABLE\_LOGIN ", "

TABLE\_PSSWD ") "

"VALUES (:Login, :Password)");

query.bindValue(":Login", data[0].toString());

query.bindValue(":Password", data[1].toString());

if(!query.exec()){

qDebug() << "error insert into " << TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoMainTableWithID(const QVariantList &data)

{

this->connectToDataBase();

QSqlQuery query;

query.prepare("INSERT INTO " TABLE " ( " TABLE\_LOGIN ", "

TABLE\_PSSWD ") "

"VALUES (:ID, :Login, :Password)");

query.bindValue(":ID", data[0].toInt());

query.bindValue(":Login", data[1].toString());

query.bindValue(":Password", data[2].toString());

if(!query.exec()){

qDebug() << "error insert into " << TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoMainTableWithKnownID(const QVariantList &data, int ID)

{

this->connectToDataBase();

if(ID != 1){

QSqlQuery query;

query.prepare("INSERT INTO " TABLE " (ID, " TABLE\_LOGIN ", " TABLE\_PSSWD ")"

"VALUES (:ID, :Login, :Password)");

query.bindValue(":ID", ID);

query.bindValue(":Login", data[0].toString());

query.bindValue(":Password", data[1].toString());

query.exec();

if(!query.exec()){

qDebug() << "error insert into " << TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("ID не может быть равным 1!");

msgBox.*exec*();

}

return false;

}

bool DataBase::checkingForIDInWeatherDB(int ID){

this->connectToWeatherDataBase();

QString sqlQuery = QString("SELECT " TABLE\_WEEKDAY " FROM "

TABLE\_WEATHER " WHERE ID = '%1'").arg(ID);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

if(query.value(0) != ID)

return true;

else return false;

}

bool DataBase::checkingForWeekdayInWeatherDB(QString weekday){

this->connectToWeatherDataBase();

QString sqlQuery = QString("SELECT " TABLE\_WEEKDAY " FROM "

TABLE\_WEATHER " WHERE " TABLE\_WEEKDAY " = '%1'").arg(weekday);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

if(query.value(0) != weekday)

return true;

else return false;

}

bool DataBase::checkingForID(int ID){

this->connectToDataBase();

QString sqlQuery = QString("SELECT " TABLE\_LOGIN " FROM "

TABLE " WHERE ID = '%1'").arg(ID);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

if(query.value(0) != ID)

return true;

else return false;

}

bool DataBase::checkingForRecord(QString login){

this->connectToDataBase();

QString sqlQuery = QString("SELECT " TABLE\_LOGIN " FROM "

TABLE " WHERE " TABLE\_LOGIN " = '%1'").arg(login);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

if(query.value(0) != login)

return true;

else return false;

}

bool DataBase::comparisonMethodIDWithLogin(int ID, QString login){

this->connectToDataBase();

QString sqlQuery = QString("SELECT ID, " TABLE\_LOGIN " FROM "

TABLE " WHERE " TABLE\_LOGIN " = '%1'").arg(login);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

if(ID == query.value(0).toInt()){

return true;

}

else return false;

return false;

}

QString DataBase::checkingForRecordReturningPsswd(QString login){

this->connectToDataBase();

QString sqlQuery = QString("SELECT " TABLE\_LOGIN " , " TABLE\_PSSWD " FROM "

TABLE " WHERE " TABLE\_LOGIN " = '%1'").arg(login);

QSqlQuery query;

query.prepare(sqlQuery);

query.exec();

query.next();

return query.value(1).toString();

}

void DataBase::deleteUserByID(int ID){

this->connectToDataBase();

QSqlQuery query;

QString sqlQuery = QString("DELETE FROM " TABLE

" WHERE ID = '%1'").arg(ID);

query.prepare(sqlQuery);

query.exec();

}

void DataBase::deleteUserByLogin(QString login){

this->connectToDataBase();

QSqlQuery query;

QString sqlQuery = QString("DELETE FROM " TABLE

" WHERE " TABLE\_LOGIN " = '%1'")

.arg(login);

query.prepare(sqlQuery);

query.exec();

}

void DataBase::deleteUserByIDAndLogin(int ID, QString login){

this->connectToDataBase();

QSqlQuery query;

QString sqlQuery = QString("DELETE FROM " TABLE

" WHERE ID = '%1' AND " TABLE\_LOGIN " = '%2'")

.arg(ID).arg(login);

query.prepare(sqlQuery);

query.exec();

}

void DataBase::connectToWeatherDataBase()

{

if(!QFile("/home/vladislav/Documents/StudyDocs/3 semester/"

"IT/Lab3/DataBaseOfUsers/DB/" DATABASE\_WEATHER\_NAME).exists()){

this->restoreWeatherDataBase();

} else {

this->openWeatherDataBase();

}

}

bool DataBase::restoreWeatherDataBase()

{

if(this->openWeatherDataBase()){

if((!this->createWeatherTable())){

return false;

} else {

return true;

}

} else {

return false;

}

return false;

}

bool DataBase::openWeatherDataBase()

{

weather\_db = QSqlDatabase::addDatabase("QSQLITE");

weather\_db.setHostName(DATABASE\_HOSTNAME\_WEATHER);

weather\_db.setDatabaseName("/home/vladislav/Documents/StudyDocs/3 semester"

"/IT/Lab3/DataBaseOfUsers/DB/" DATABASE\_WEATHER\_NAME);

if(weather\_db.open()){

return true;

} else {

return false;

}

return false;

}

bool DataBase::createWeatherTable()

{

this->connectToWeatherDataBase();

QSqlQuery query;

if(!query.exec("CREATE TABLE " TABLE\_WEATHER

"(ID INTEGER PRIMARY KEY AUTOINCREMENT, "

"Weekday DATE NOT NULL,"

"Date\_ DATE NOT NULL, "

"Month DATE NOT NULL, "

"Label TEXT NOT NULL, "

"Temperature TEXT NOT NULL)")){

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoWeatherTable(const QVariantList &data){

this->connectToWeatherDataBase();

QSqlQuery query;

query.prepare("INSERT INTO " TABLE\_WEATHER " ( " TABLE\_WEEKDAY ", "

TABLE\_DATE ", " TABLE\_MONTH ", "

TABLE\_LABEL ", " TABLE\_TEMPERATURE ")"

" VALUES (:Weekday, :Date\_, :Month, :Label, :Temperature)");

query.bindValue(":Weekday", data[0].toString());

query.bindValue(":Date\_", data[1].toString());

query.bindValue(":Month", data[2].toString());

query.bindValue(":Label", data[3].toString());

query.bindValue(":Temperature", data[4].toString());

if(!query.exec()){

qDebug() << "error insert into " << TABLE\_WEATHER;

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoWeatherTableWithID(const QVariantList &data, int ID){

this->connectToWeatherDataBase();

QSqlQuery query;

query.prepare("INSERT INTO " TABLE\_WEATHER " (ID, " TABLE\_WEEKDAY ", "

TABLE\_DATE ", " TABLE\_MONTH ", "

TABLE\_LABEL ", " TABLE\_TEMPERATURE ")"

" VALUES (:ID, :Weekday, :Date\_, :Month, :Label, :Temperature)");

query.bindValue(":ID", ID);

query.bindValue(":Weekday", data[0].toString());

query.bindValue(":Date\_", data[1].toString());

query.bindValue(":Month", data[2].toString());

query.bindValue(":Label", data[3].toString());

query.bindValue(":Temperature", data[4].toString());

query.exec();

if(!query.exec()){

qDebug() << "error insert into " << TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DataBase::insertIntoWeatherTableWithKnownID(const QVariantList &data, int ID)

{

this->connectToWeatherDataBase();

QSqlQuery query;

query.prepare("INSERT INTO " TABLE\_WEATHER " (ID, " TABLE\_WEEKDAY ", "

TABLE\_DATE ", " TABLE\_MONTH ", "

TABLE\_LABEL ", " TABLE\_TEMPERATURE ")"

" VALUES (:ID, :Weekday, :Date\_, :Month, :Label, :Temperature)");

query.bindValue(":ID", ID);

query.bindValue(":Weekday", data[0].toString());

query.bindValue(":Date\_", data[1].toString());

query.bindValue(":Month", data[2].toString());

query.bindValue(":Label", data[3].toString());

query.bindValue(":Temperature", data[4].toString());

query.exec();

if(!query.exec()){

qDebug() << "error insert into " << TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

void DataBase::deleteWeatherRecordByID(int ID){

this->connectToWeatherDataBase();

QSqlQuery query;

QString sqlQuery = QString("DELETE FROM " TABLE\_WEATHER

" WHERE ID = '%1'").arg(ID);

query.prepare(sqlQuery);

query.exec();

}

void DataBase::deleteAllWeatherRecords(){

this->connectToWeatherDataBase();

QSqlQuery query;

query.exec("DELETE FROM " TABLE\_WEATHER);

query.exec("VACUUM");

query.exec("DROP TABLE " TABLE\_WEATHER);

}

void DataBase::modifyWeatherRecordWithID(const QVariantList &data, int ID){

this->connectToWeatherDataBase();

QSqlQuery query;

QString sqlQuery = QString("UPDATE " TABLE\_WEATHER

" SET " TABLE\_WEEKDAY " = '%1', "

TABLE\_DATE " = '%2', "

TABLE\_MONTH " = '%3', "

TABLE\_LABEL " = '%4', "

TABLE\_TEMPERATURE " = '%5' "

" WHERE ID = %6;")

.arg(data[0].toString())

.arg(data[1].toString())

.arg(data[2].toString())

.arg(data[3].toString())

.arg(data[4].toString())

.arg(ID);

query.prepare(sqlQuery);

query.exec();

}

int DataBase::checkUsersDataBaseCountOfRecords(){

this->connectToDataBase();

QSqlQuery query;

QString sqlQuery = QString("SELECT COUNT(\*) "

" FROM " TABLE);

query.prepare(sqlQuery);

query.exec();

query.next();

return query.value(0).toInt();

}

int DataBase::checkWeatherDataBaseCountOfRecords(){

this->connectToWeatherDataBase();

QSqlQuery query;

QString sqlQuery = QString("SELECT COUNT(\*) "

" FROM " TABLE\_WEATHER);

query.prepare(sqlQuery);

query.exec();

query.next();

return query.value(0).toInt();

}

int DataBase::checkWeatherDataBaseCountOfWeekdayRecords(QString weekday){

this->connectToWeatherDataBase();

QSqlQuery query;

QString sqlQuery = QString("SELECT COUNT(\*) "

" FROM " TABLE\_WEATHER

" WHERE " TABLE\_WEEKDAY " = '%1'").arg(weekday);

query.prepare(sqlQuery);

query.exec();

query.next();

return query.value(0).toInt();

}

QVariantList DataBaseReadMethods::readDataFromUsersSqlFileToQVariantList(){

QVariantList data;

this->connectToDataBase();

int countOfRecords = this->checkUsersDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString("SELECT \* "

"FROM " TABLE);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 3; i++){

if(query.value(i + 1).toString() != nullptr)

data.append(query.value(i + 1));

}

query.next();

}

return data;

}

QVariantList DataBaseReadMethods::readDataFromWeatherSqlFileToQVariantList(){

QVariantList data;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString("SELECT \* "

"FROM " TABLE\_WEATHER);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

data.append(query.value(i + 1));

}

query.next();

}

return data;

}

QVariantList DataBaseReadMethods::readWeekdayFromWeatherSqlFileToQVariantList(QString weekday){

QVariantList data;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString("SELECT \* "

"FROM " TABLE\_WEATHER

" WHERE " TABLE\_WEEKDAY " = '%1'").arg(weekday);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

data.append(query.value(i + 1));

}

query.next();

}

return data;

}

QVariantList DataBaseReadMethods::readDateFromWeatherSqlFileToQVariantList(QString date){

QVariantList data;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString("SELECT \* "

"FROM " TABLE\_WEATHER

" WHERE " TABLE\_DATE " = '%1'").arg(date);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

data.append(query.value(i + 1));

}

query.next();

}

return data;

}

QVariantList DataBaseReadMethods::readMonthFromWeatherSqlFileToQVariantList(QString month){

QVariantList data;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString("SELECT \* "

"FROM " TABLE\_WEATHER

" WHERE " TABLE\_MONTH " = '%1'").arg(month);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

data.append(query.value(i + 1));

}

query.next();

}

return data;

}

void DataBase::backupDataFromWeatherDB(){

this->connectToWeatherDataBase();

DataBaseReadMethods \*readFromDb = new DataBaseReadMethods();

QVariantList savedData;

savedData = readFromDb->readDataFromWeatherSqlFileToQVariantList();

setSaved(savedData);

}

void DataBase::undoMethod(){

this->deleteAllWeatherRecords();

this->createWeatherTable();

this->connectToWeatherDataBase();

DataBaseReadMethods \*readFromDb = new DataBaseReadMethods();

QVariantList savedCopy = getSaved();

for(int i = 1; i <= saved.size(); i++){

savedCopy.append(saved.value(i - 1));

if(i % 5 == 0){

readFromDb->insertIntoWeatherTable(savedCopy);

savedCopy.clear();

}

}

QMessageBox msgBox;

msgBox.setWindowTitle("Отмена действий");

msgBox.setText("Отмена предыдущих действий произошла успешно!");

msgBox.*exec*();

}

void DataBase::searchByID(int ID){

this->backupDataFromWeatherDB();

QVariantList requiredLine;

this->connectToWeatherDataBase();

QSqlQuery query;

QString sqlQuery = QString("SELECT " TABLE\_WEEKDAY ", " TABLE\_DATE ", " TABLE\_MONTH ", "

TABLE\_LABEL ", " TABLE\_TEMPERATURE

" FROM " TABLE\_WEATHER

" WHERE ID = %1").arg(ID);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int i = 0; i < 5; i++){

requiredLine.append(query.value(i));

}

this->deleteAllWeatherRecords();

this->createWeatherTable();

this->insertIntoWeatherTableWithID(requiredLine, ID);

requiredLine.clear();

}

void DataBase::searchByWeekday(QString weekday){

DataBaseReadMethods \*read = new DataBaseReadMethods();

QVariantList weekdayData = read->readWeekdayFromWeatherSqlFileToQVariantList(weekday);

QVariantList copyData;

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

for(int i = 1; i <= saved.size(); i++){

copyData.append(weekdayData.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

weekdayData.clear();

}

void DataBase::searchByDate(QString date){

DataBaseReadMethods \*read = new DataBaseReadMethods();

QVariantList dateData = read->readDateFromWeatherSqlFileToQVariantList(date);

QVariantList copyData;

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

for(int i = 1; i <= saved.size(); i++){

copyData.append(dateData.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

dateData.clear();

}

void DataBase::searchByMonth(QString month){

DataBaseReadMethods \*read = new DataBaseReadMethods();

QVariantList dateData = read->readMonthFromWeatherSqlFileToQVariantList(month);

QVariantList copyData;

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

for(int i = 1; i <= saved.size(); i++){

copyData.append(dateData.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

dateData.clear();

}

void DataBase::setSaved(QVariantList savedData){

saved = savedData;

}

QVariantList DataBase::getSaved(){

return saved;

}

void SortMethods::sortingByWeekday(){

QVariantList sortedDataByWeekday;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString(" SELECT \* "

" FROM " TABLE\_WEATHER

" ORDER BY " TABLE\_WEEKDAY);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

sortedDataByWeekday.append(query.value(i + 1));

}

query.next();

}

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

QVariantList copyData;

for(int i = 1; i <= saved.size(); i++){

copyData.append(sortedDataByWeekday.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

sortedDataByWeekday.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Сортировка по дням недели");

msgBox.setText("Данные были отсортированы по дням недели.");

msgBox.*exec*();

}

void SortMethods::sortingByDate(){

QVariantList sortedDataByDate;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString(" SELECT \* "

" FROM " TABLE\_WEATHER

" ORDER BY " TABLE\_DATE);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

sortedDataByDate.append(query.value(i + 1));

}

query.next();

}

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

QVariantList copyData;

for(int i = 1; i <= saved.size(); i++){

copyData.append(sortedDataByDate.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

sortedDataByDate.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Сортировка по дате");

msgBox.setText("Данные были отсортированы по дате.");

msgBox.*exec*();

}

void SortMethods::sortingByMonth(){

QVariantList sortedDataByMonth;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString(" SELECT \* "

" FROM " TABLE\_WEATHER

" ORDER BY " TABLE\_MONTH);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

sortedDataByMonth.append(query.value(i + 1));

}

query.next();

}

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

QVariantList copyData;

for(int i = 1; i <= saved.size(); i++){

copyData.append(sortedDataByMonth.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

sortedDataByMonth.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Сортировка по месяцу");

msgBox.setText("Данные были отсортированы по месяцу.");

msgBox.*exec*();

}

void SortMethods::sortingByGeneralCharacteristics(){

QVariantList sortedDataByGeneralCharacteristics;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString(" SELECT \* "

" FROM " TABLE\_WEATHER

" ORDER BY " TABLE\_LABEL);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

sortedDataByGeneralCharacteristics.append(query.value(i + 1));

}

query.next();

}

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

QVariantList copyData;

for(int i = 1; i <= saved.size(); i++){

copyData.append(sortedDataByGeneralCharacteristics.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

sortedDataByGeneralCharacteristics.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Сортировка по характеристикам");

msgBox.setText("Данные были отсортированы по основным характеристикам.");

msgBox.*exec*();

}

void SortMethods::sortingByTemperature(){

QVariantList sortedDataByTempreature;

this->connectToWeatherDataBase();

int countOfRecords = this->checkWeatherDataBaseCountOfRecords();

QSqlQuery query;

QString sqlQuery = QString(" SELECT \* "

" FROM " TABLE\_WEATHER

" ORDER BY " TABLE\_TEMPERATURE);

query.prepare(sqlQuery);

query.exec();

query.next();

for(int j = 0; j < countOfRecords; j++){

for(int i = 0; i < 6; i++){

if(query.value(i + 1).toString() != nullptr)

sortedDataByTempreature.append(query.value(i + 1));

}

query.next();

}

this->backupDataFromWeatherDB();

this->deleteAllWeatherRecords();

this->createWeatherTable();

QVariantList copyData;

for(int i = 1; i <= saved.size(); i++){

copyData.append(sortedDataByTempreature.value(i - 1));

if(i % 5 == 0){

this->insertIntoWeatherTable(copyData);

copyData.clear();

}

}

sortedDataByTempreature.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Сортировка по температуре");

msgBox.setText("Данные были отсортированы по температуре.");

msgBox.*exec*();}

**3.1 Код файла deleteuserfromdb.h:**

#ifndef DELETEUSERFROMDB\_H

#define DELETEUSERFROMDB\_H

#include <QWidget>

#include <QSqlQuery>

#include <QMessageBox>

#include "database.h"

namespace Ui {

class deleteUserFromDB;

}

class deleteUserFromDB : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit deleteUserFromDB(QWidget \*parent = nullptr);

~*deleteUserFromDB*();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_loginCheckBox\_stateChanged(int arg1);

private:

Ui::deleteUserFromDB \*ui;

DataBase \*db;

};

#endif // DELETEUSERFROMDB\_H

**3.2 Код файла deleteuserfromdb.cpp:**

#include "deleteuserfromdb.h"

#include "ui\_deleteuserfromdb.h"

deleteUserFromDB::deleteUserFromDB(QWidget \*parent) :

QWidget(parent),

ui(new Ui::deleteUserFromDB)

{

ui->setupUi(this);

db = new DataBase();

}

deleteUserFromDB::~deleteUserFromDB()

{

delete ui;

}

void deleteUserFromDB::on\_pushButton\_clicked()

{

if(db->checkingForID(ui->IDLineEdit->text().toInt())){

if(ui->deleteLoginLineEdit->text() != nullptr){

db->deleteUserByLogin(ui->deleteLoginLineEdit->text());

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление");

msgBox.setText("Пользователь успешно удален.");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else if(ui->IDLineEdit->text() != nullptr){

if(ui->IDLineEdit->text().toInt() != 1){

db->deleteUserByID(ui->IDLineEdit->text().toInt());

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление");

msgBox.setText("Пользователь успешно удален.");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else if(ui->IDLineEdit->text().toInt() == 1){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Невозможно удалить запись ID которого равен 1!");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка");

}

}

else if(ui->deleteLoginLineEdit->text() != nullptr && ui->IDLineEdit->text() != nullptr){

if(ui->IDLineEdit->text().toInt() != 1){

if(db->comparisonMethodIDWithLogin(ui->IDLineEdit->text().toInt(),

ui->deleteLoginLineEdit->text())){

db->deleteUserByIDAndLogin(ui->IDLineEdit->text().toInt(),

ui->deleteLoginLineEdit->text());

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление");

msgBox.setText("Пользователь успешно удален.");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else if(!db->comparisonMethodIDWithLogin(ui->IDLineEdit->text().toInt(),

ui->deleteLoginLineEdit->text())){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("ID и логин не совпадают с записью.");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка");

}

}

else if(ui->IDLineEdit->text().toInt() == 1){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Невозможно удалить запись ID которого равен 1!");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка");

}

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка");

}

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка");

}

this->close();

}

void deleteUserFromDB::on\_pushButton\_2\_clicked()

{

db->connectToDataBase();

QSqlQuery query;

query.exec("DROP TABLE " TABLE);

db->createMainTable();

db->closeDataBase();

QVariantList data;

data.append("admin");

data.append("admin");

db->insertIntoMainTable(data);

data.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление всех записей");

msgBox.setText("Все записи успешно удалены, за исключением записи,"

" содержащей в себе привилегилированного пользователя.");

msgBox.exec();

ui->deleteLoginLineEdit->clear();

ui->IDLineEdit->clear();

this->close();

}

**4.1 Код файла deleteweatherrecord.h:**

#ifndef DELETEWEATHERRECORD\_H

#define DELETEWEATHERRECORD\_H

#include <QWidget>

#include <QSqlQuery>

#include <QMessageBox>

#include <QLineEdit>

#include "database.h"

namespace Ui {

class DeleteWeatherRecord;

}

class DeleteWeatherRecord : public QWidget

{

Q\_OBJECT

public:

explicit DeleteWeatherRecord(QWidget \*parent = nullptr);

~DeleteWeatherRecord();

signals:

void showWindow();

private slots:

void on\_deletePushButton\_clicked();

void on\_deleteAllPushButton\_clicked();

private:

Ui::DeleteWeatherRecord \*ui;

DataBase \*db;

};

#endif // DELETEWEATHERRECORD\_H

**4.2 Код файла deleteweatherrecord.cpp:**

#include "deleteweatherrecord.h"

#include "ui\_deleteweatherrecord.h"

DeleteWeatherRecord::DeleteWeatherRecord(QWidget \*parent) :

QWidget(parent),

ui(new Ui::DeleteWeatherRecord)

{

ui->setupUi(this);

db = new DataBase();

}

DeleteWeatherRecord::~DeleteWeatherRecord()

{

delete ui;

}

void DeleteWeatherRecord::on\_deletePushButton\_clicked()

{

int id = ui->IDLineEdit->text().toInt();

if((id > 0 && id <= db->checkWeatherDataBaseCountOfRecords()) ||

db->checkingForIDInWeatherDB(ui->IDLineEdit->text().toInt())){

if(ui->IDLineEdit->text() != nullptr){

db->deleteWeatherRecordByID(id);

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление");

msgBox.setText("Запись была успешно удалена.");

msgBox.*exec*();

}

else if(ui->IDLineEdit->text() == nullptr){

QMessageBox::warning(this, "Внимание", "ID не введено");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else if(id > db->checkWeatherDataBaseCountOfRecords()){

QMessageBox::warning(this, "Внимание", "Некорректно введен ID.");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

ui->IDLineEdit->clear();

this->close();

}

void DeleteWeatherRecord::on\_deleteAllPushButton\_clicked()

{

db->deleteAllWeatherRecords();

db->createWeatherTable();

QMessageBox msgBox;

msgBox.setWindowTitle("Удаление всех записей");

msgBox.setText("Все записи успешно удалены.");

msgBox.exec();

this->close();

}

**5.1 Код файла htmlParser.h:**

#ifndef HTMLPARSER\_H

#define HTMLPARSER\_H

#define BUFF\_SIZE 256

#include <QVariantList>

#include <cstdlib>

#include <fstream>

#include <ostream>

#include <iostream>

#include <cctype>

#include <cstring>

#include <sys/stat.h>

#include <gumbo.h>

#include "maindatabase.h"

class Sinoptik{

public:

struct sinoptik{

char \*weekDay;

char \*date;

char \*month;

char \*label;

char \*temperature;

};

struct sinoptik weather[BUFF\_SIZE];

public:

explicit Sinoptik();

virtual ~*Sinoptik*();

void findTemperature(GumboNode \*node);

void findLabel(GumboNode \*node, const char \*lostLabel);

void findWeekDay(GumboNode \*node, const char \*lostWeekDay, const char \*curDay);

void findDate(GumboNode \*node, const char \*lost);

void findMonth(GumboNode \*node, const char \*lost);

int htmlParse();

void fillWeatherDB();

};

#endif // HTMLPARSER\_H

**5.2 Код файла htmlParser.cpp:**

#include "htmlParser.h"

int counter = 0, k = 0, j = 0, z = 0, a = 0;

Sinoptik::Sinoptik(){

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::in);

}

Sinoptik::~*Sinoptik*(){

}

void Sinoptik::findTemperature(GumboNode \*node){

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

if(node->type != *GUMBO\_NODE\_ELEMENT*)

return;

if(node->v.element.tag == *GUMBO\_TAG\_B*){

GumboNode \*textBetweenTagB = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagB->type == *GUMBO\_NODE\_TEXT* || textBetweenTagB->type != *GUMBO\_NODE\_WHITESPACE*){

if(z % 2 != 0){

#ifdef PRNT

printf("max = %s\n", textBetweenTagB->v.text.text);

#endif

weather[z].temperature = (char \*)calloc(strlen(textBetweenTagB->v.text.text), sizeof(char \*));

strcpy(weather[z].temperature, (char \*)textBetweenTagB->v.text.text);

copyFile << weather[z].temperature << " ";

}

else{

#ifdef PRNT

printf("min = %s\n", textBetweenTagB->v.text.text);

#endif

weather[z].temperature = (char \*)calloc(strlen(textBetweenTagB->v.text.text), sizeof(char \*));

strcpy(weather[z].temperature, (char \*)textBetweenTagB->v.text.text);

copyFile << weather[z].temperature << "/";

}

z++;

}

}

GumboVector \*child = &node->v.element.children;

for(size\_t i = 0; i < (child->length); i++){

findTemperature((GumboNode \*)(child->data[i]));

if(z == 14){

i = child->length;

}

}

copyFile.close();

}

void Sinoptik::findLabel(GumboNode \*node, const char \*lostLabel){

GumboAttribute \*className;

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

if(node->type != *GUMBO\_NODE\_ELEMENT*)

return;

if(node->v.element.tag == *GUMBO\_TAG\_LABEL* &&

(className = gumbo\_get\_attribute(&node->v.element.attributes, "class")) &&

!strcmp(className->value, lostLabel)){

GumboNode \*textBetweenTagLabel = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagLabel->type == *GUMBO\_NODE\_TEXT*

|| textBetweenTagLabel->type != *GUMBO\_NODE\_WHITESPACE*){

std::string tmp = (char \*)calloc(strlen(textBetweenTagLabel->v.text.text), sizeof(char \*));

tmp = textBetweenTagLabel->v.text.text;

for(size\_t v = 0; v < tmp.size(); v++){

if(tmp[v] == ' ')

tmp[v] = '\_';

}

#ifdef PRNT

printf("%s\n", textBetweenTagLabel->v.text.text);

#endif

if(tmp[0] == '\n'){

tmp[0] = ' ';

}

for(size\_t v = 0; v < tmp.size(); v++){

if(tmp[v] == ',' && tmp[v + 1] == ' '){

tmp.erase(v, 1);

tmp[v + 1] = ' ';

}

}

if(j == 6){

tmp[tmp.size()] = ' ';

}

weather[j].label = (char \*)calloc(strlen(textBetweenTagLabel->v.text.text), sizeof(char \*));

strcpy(weather[j].label, tmp.c\_str());

copyFile << weather[j].label;

}

j++;

}

GumboVector \*child = &node->v.element.children;

for(size\_t i = 0; i < (child->length); i++){

findLabel((GumboNode \*)(child->data[i]), lostLabel);

if(j == 7){

i = child->length;

}

}

copyFile.close();

}

void Sinoptik::findWeekDay(GumboNode \*node, const char \*lostWeekDay, const char \*curDay){

GumboAttribute \*className;

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

if(node->type != *GUMBO\_NODE\_ELEMENT*)

return;

if(node->v.element.tag == *GUMBO\_TAG\_P* &&

(className = gumbo\_get\_attribute(&node->v.element.attributes, "class")) &&

!strcmp(className->value, curDay)){

GumboNode \*textBetweenTagP = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagP->type == *GUMBO\_NODE\_TEXT* || textBetweenTagP->type != *GUMBO\_NODE\_WHITESPACE*){

#ifdef PRNT

printf("%s\n", textBetweenTagP->v.text.text);

#endif

weather[a].weekDay = (char \*)calloc(strlen(textBetweenTagP->v.text.text), sizeof(char \*));

if(weather[a].weekDay != NULL && (a == 0 || a == 1)){

strcpy(weather[a].weekDay, textBetweenTagP->v.text.text);

copyFile << weather[a].weekDay << " ";

a++;

}

}

}

if(node->v.element.tag == *GUMBO\_TAG\_A* &&

(className = gumbo\_get\_attribute(&node->v.element.attributes, "class")) &&

!strcmp(className->value, lostWeekDay)){

GumboNode \*textBetweenTagA = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagA->type == *GUMBO\_NODE\_TEXT* || textBetweenTagA->type != *GUMBO\_NODE\_WHITESPACE*){

#ifdef PRNT

printf("%s\n", textBetweenTagA->v.text.text);

#endif

weather[k+a].weekDay = (char \*)calloc(strlen(textBetweenTagA->v.text.text), sizeof(char \*));

strcpy(weather[k+a].weekDay, textBetweenTagA->v.text.text);

copyFile << weather[k+a].weekDay << " ";

}

k++;

counter++;

}

GumboVector \*child = &node->v.element.children;

for(size\_t i = 0; i < (child->length); i++){

findWeekDay((GumboNode \*)(child->data[i]), lostWeekDay, curDay);

if(counter == 7){

i = child->length;

}

}

copyFile.close();

}

int t = 0, f = 0;

void Sinoptik::findDate(GumboNode \*node, const char \*lost){

GumboAttribute \*className;

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

if(node->type != *GUMBO\_NODE\_ELEMENT*)

return;

if(node->v.element.tag == *GUMBO\_TAG\_P* &&

(className = gumbo\_get\_attribute(&node->v.element.attributes, "class")) &&

!strcmp(className->value, lost)){

GumboNode \*textBetweenTagP = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagP->type == *GUMBO\_NODE\_TEXT* || textBetweenTagP->type != *GUMBO\_NODE\_WHITESPACE*){

#ifdef PRNT

printf("%s\n", textBetweenTagP->v.text.text);

#endif

weather[t].date = (char \*)calloc(strlen(textBetweenTagP->v.text.text), sizeof(char \*));

strcpy(weather[t].date, textBetweenTagP->v.text.text);

if(t != 7)

copyFile << weather[t].date << " ";

}

t++;

}

GumboVector \*child = &node->v.element.children;

for(size\_t i = 0; i < (child->length); i++){

findDate((GumboNode \*)(child->data[i]), lost);

}

copyFile.close();

}

void Sinoptik::findMonth(GumboNode \*node, const char \*lost){

GumboAttribute \*className;

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

if(node->type != *GUMBO\_NODE\_ELEMENT*)

return;

if(node->v.element.tag == *GUMBO\_TAG\_P* &&

(className = gumbo\_get\_attribute(&node->v.element.attributes, "class")) &&

!strcmp(className->value, lost)){

GumboNode \*textBetweenTagP = (GumboNode \*)(node->v.element.children.data[0]);

if(textBetweenTagP->type == *GUMBO\_NODE\_TEXT* || textBetweenTagP->type != *GUMBO\_NODE\_WHITESPACE*){

#ifdef PRNT

printf("%s\n", textBetweenTagP->v.text.text);

#endif

weather[f].month = (char \*)calloc(strlen(textBetweenTagP->v.text.text), sizeof(char \*));

strcpy(weather[f].month, textBetweenTagP->v.text.text);

if(f != 7)

copyFile << weather[f].month << " ";

}

f++;

}

GumboVector \*child = &node->v.element.children;

for(size\_t i = 0; i < (child->length); i++){

findMonth((GumboNode \*)(child->data[i]), lost);

}

copyFile.close();

}

void Sinoptik::fillWeatherDB(){

QVariantList saved;

MainDataBase \*db;

DataBaseReadMethods \*readFromDb = new DataBaseReadMethods();

db = new MainDataBase;

db->connectToDataBase();

QSqlQuery query;

std::fstream file;

std::string str;

int countOfSpaces = 0;

file.open("DataBase.txt", std::ios\_base::in);

if(file.is\_open()){

while(getline(file, *str*)){}

}

for(size\_t i = 0 ; i < str.length(); i++){

if(str[i] == ' ' && str[i + 1] == ' ') {

k++;

str.erase(i, 1);

i--;

}

if(str[i] == ' '){

countOfSpaces++;

if(countOfSpaces % 7 == 0){

str[i] = '\n';

}

}

}

file.close();

std::ofstream ofs;

ofs.open("DataBase.txt", std::ofstream::out | std::ofstream::trunc);

ofs.close();

std::ofstream copyFile;

copyFile.open("DataBase.txt", std::ios\_base::app);

copyFile << str;

copyFile.close();

QVariantList data;

std::ifstream stream;

std::string weekday[7], date[7], month[7], label[7], temperature[7];

QVariant weekday\_dest[7], date\_dest[7], month\_dest[7], label\_dest[7], temperature\_dest[7];

stream.open("DataBase.txt", std::ifstream::in);

for(int i = 0; i < 7; i++){

stream >> weekday[i];

weekday\_dest[i] = QString::fromStdString(weekday[i]);

}

for(int i = 0; i < 7; i++){

stream >> date[i];

date\_dest[i] = QString::fromStdString(date[i]);

}

for(int i = 0; i < 7; i++){

stream >> month[i];

month\_dest[i] = QString::fromStdString(month[i]);

}

for(int i = 0; i < 7; i++){

stream >> label[i];

label\_dest[i] = QString::fromStdString(label[i]);

}

for(int i = 0; i < 7; i++){

stream >> temperature[i];

temperature\_dest[i] = QString::fromStdString(temperature[i]);

}

if(!db->mainDataBaseNotEmpty()){

for(int i = 0; i < 7; i++){

data.append(weekday\_dest[i]);

data.append(date\_dest[i]);

data.append(month\_dest[i]);

data.append(label\_dest[i]);

data.append(temperature\_dest[i]);

db->insertIntoMainTable(data);

data.clear();

}

}

else{

saved = readFromDb->readDataFromWeatherSqlFileToQVariantList();

QVariantList savedCopy;

readFromDb->deleteAllWeatherRecords();

readFromDb->createWeatherTable();

for(int i = 1; i <= saved.size(); i++){

savedCopy.append(saved.value(i - 1));

if(i % 5 == 0){

readFromDb->insertIntoWeatherTable(savedCopy);

savedCopy.clear();

}

}

}

saved.clear();

stream.close();

delete db;

}

int Sinoptik::htmlParse(){

const char \*fileName = "weather.html";

FILE \*fin;

char \*buffer = (char\*)calloc(BUFF\_SIZE, sizeof(char\*));

char \*copy;

long bytes = 0L;

if((fin = fopen(fileName, "rb")) == NULL){

perror("Error: Can't read file. Make sure of existanse this file.");

return EXIT\_FAILURE;

}

fseek(fin, 0L, SEEK\_END);

bytes = ftell(fin);

fseek(fin, 0L, SEEK\_SET);

copy = (char\*)calloc(bytes, sizeof(char));

fseek(fin, 0L, SEEK\_SET);

fread(copy, sizeof(char\*), bytes, fin);

#ifdef PRINT\_HTML

printf("\n%s\n", copy);

printf("\nSuccessfully read %ld bytes\n", bytes);

#endif

free(buffer);

const char \*curDay = "weather\_\_content\_tab-day";

const char \*lostWeekDay = "weather\_\_content\_tab\_a";

const char \*lostDay = "weather\_\_content\_tab-date day\_red";

const char \*lostMonth = "weather\_\_content\_tab-month";

const char \*lostLabel = "show-tooltip";

GumboOutput \*out = gumbo\_parse(copy);

findWeekDay(out->root, lostWeekDay, curDay);

findDate(out->root, lostDay);

findMonth(out->root, lostMonth);

findLabel(out->root, lostLabel);

findTemperature(out->root);

gumbo\_destroy\_output(&kGumboDefaultOptions, out);

free(copy);

fclose(fin);

system(qPrintable("clear"));

return EXIT\_SUCCESS;

}

**6.1 Код файла htmlThief.h:**

#ifndef HTMLTHIEF\_H

#define HTMLTHIEF\_H

#include <c++/10/curl/curl.h>

#include <istream>

#include <cstdlib>

static size\_t writeData(void\* ptr, size\_t size, size\_t nmemb, void\* stream){

size\_t written = fwrite(ptr, size, nmemb, (FILE\*)stream);

return written;

}

int htmlThief(int argc, char\* argv[]){

CURL\* curlHandle;

const char\* pageFileName = "weather.html";

FILE\* pageFile;

if(argc < 2){

printf("Usage: %s<URL>\n", argv[0]);

return EXIT\_FAILURE;

}

curl\_global\_init(CURL\_GLOBAL\_ALL); // flag CURL\_GLOBAL\_ALL -

// initializing everything is possible (http, ssl, etc.)

curlHandle = curl\_easy\_init(); //initializing the curl session

curl\_easy\_setopt(curlHandle, *CURLOPT\_URL*, argv[1]); //setting up the URL

curl\_easy\_setopt(curlHandle, *CURLOPT\_VERBOSE*, 1L); //switch on full protocol/debug output while test

curl\_easy\_setopt(curlHandle, *CURLOPT\_NOPROGRESS*, 0L); //enable progress bar

curl\_easy\_setopt(curlHandle, *CURLOPT\_WRITEFUNCTION*, writeData);

if((pageFile = (fopen(pageFileName, "wb"))) != NULL){

curl\_easy\_setopt(curlHandle, *CURLOPT\_WRITEDATA*, pageFile);

curl\_easy\_perform(curlHandle); //perform the transfer as descripted in options

fclose(pageFile);

}

else

perror("Error write data in file");

curl\_easy\_cleanup(curlHandle);

curl\_global\_cleanup();

return EXIT\_SUCCESS;

}

#endif // HTMLTHIEF\_H

**7.1 Код файла loginning.h:**

#ifndef LOGINNING\_H

#define LOGINNING\_H

#include <QDialog>

#include <QMessageBox>

#include "database.h"

#include "maindatabase.h"

namespace Ui {

class loginning;

}

class loginning : public QDialog

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit loginning(QWidget \*parent = nullptr);

virtual ~*loginning*();

private slots:

void on\_exitButton\_clicked();

void on\_signInButton\_clicked();

void on\_confirmationPsswdLineEdit\_editingFinished();

void on\_signUpButton\_clicked();

private:

Ui::loginning \*ui;

MainDataBase \*mainDB;

DataBase \*db;

SearchRecord \*search;

};

#endif // LOGINNING\_H

**7.2 Код файла loginning.cpp:**

#include "loginning.h"

#include "ui\_loginning.h"

loginning::loginning(QWidget \*parent) :

QDialog(parent),

ui(new Ui::loginning)

{

ui->setupUi(this);

mainDB = new MainDataBase();

connect(mainDB, &MainDataBase::showMainDB, this, &loginning::show);

db = new DataBase();

ui->confirmationPsswdLineEdit->hide();

}

loginning::~loginning()

{

delete ui;

}

void loginning::on\_exitButton\_clicked()

{

this->close();

}

void loginning::on\_signInButton\_clicked()

{

db->connectToDataBase();

QSqlQuery query;

query.exec("DROP TABLE " TABLE\_WEATHER);

MainDataBase \*weather\_db;

weather\_db = new MainDataBase;

weather\_db->createMainTable();

ui->confirmationPsswdLineEdit->hide();

QString login = ui->loginLineEdit->text();

QString psswd = ui->psswdLineEdit->text();

QString psswdFromPsswdLineEdit = db->checkingForRecordReturningPsswd(login);

if(login != nullptr || psswd != nullptr){

if(login == "admin" && psswd == "admin"){

QMessageBox msgBox;

msgBox.setWindowTitle("Вход");

msgBox.setText("Успешно выполнен вход под Admin.");

msgBox.exec();

this->close();

mainDB->show();

emit showWindow();

}

else{

if(!db->checkingForRecord(login)){

if(psswd == psswdFromPsswdLineEdit){

QMessageBox msgBox;

msgBox.setWindowTitle("Вход");

msgBox.setText("Вы успешно вошли под пользователем "

+ ui->loginLineEdit->text());

msgBox.exec();

this->close();

mainDB->getAddNewButton()->hide();

mainDB->getEncodeButton()->hide();

mainDB->getDecodeButton()->hide();

mainDB->getDeleteButton()->hide();

mainDB->getModifyButton()->hide();

mainDB->getSearchButton()->show();

mainDB->getUndoButton()->show();

mainDB->getSortingButton()->show();

mainDB->getLabel()->setText("Панель управления пользователя " +

ui->loginLineEdit->text());

mainDB->show();

}

else if(psswd != psswdFromPsswdLineEdit){

QMessageBox::warning(this, "Внимание", "Не найдено ни одного пользователя с "

"логином " +

ui->loginLineEdit->text() + "!");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else{

QMessageBox::warning(this, "Внимание", "Не найдено ни одного пользователя с "

"логином " +

ui->loginLineEdit->text() + "!");

}

}

}

}

void loginning::on\_confirmationPsswdLineEdit\_editingFinished()

{

if(ui->psswdLineEdit->text() != ui->confirmationPsswdLineEdit->text()){

ui->confirmationPsswdLineEdit->setStyleSheet("QLineEdit {"

"border: 1px solid red;"

"border-radius: 4px;"

"}");

}

}

void loginning::on\_signUpButton\_clicked()

{

ui->confirmationPsswdLineEdit->show();

QSqlQuery query;

query.exec("DROP TABLE " TABLE\_WEATHER);

MainDataBase \*weather\_db;

weather\_db = new MainDataBase;

weather\_db->createMainTable();

QString newLogin = ui->loginLineEdit->text();

QString newPsswd = nullptr;

if(ui->confirmationPsswdLineEdit->text() != nullptr){

if(db->checkingForRecord(newLogin)){

if(ui->psswdLineEdit->text() == ui->confirmationPsswdLineEdit->text()){

newPsswd = ui->psswdLineEdit->text();

QVariantList data;

data.append(newLogin);

data.append(newPsswd);

db->insertIntoMainTable(data);

data.clear();

QMessageBox msgBox;

msgBox.setWindowTitle("Регистрация");

msgBox.setText("Вы успешно зарегистрировались."

" \nВы были добавлены в базу данных пользователей.");

msgBox.exec();

}

else if(ui->psswdLineEdit->text() != ui->confirmationPsswdLineEdit->text()){

QMessageBox::warning(this, "Внимание", "Passwords don't match, please try again.");

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->confirmationPsswdLineEdit->clear();

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else{

QMessageBox::warning(this, "Внимание", "Пользователь с таким логином уже существует!");

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->confirmationPsswdLineEdit->clear();

}

}

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->confirmationPsswdLineEdit->clear();

}

**8.1 Код файла maindatabase.h:**

#ifndef MAINDATABASE\_H

#define MAINDATABASE\_H

#define DATABASE\_HOSTNAME\_WEATHER "WeatherDataBase"

#define DATABASE\_WEATHER\_NAME "weather.sql"

#define TABLE\_WEATHER "Weather"

#define TABLE\_WEEKDAY "Weekday"

#define TABLE\_DATE "Date\_"

#define TABLE\_MONTH "Month"

#define TABLE\_LABEL "Label"

#define TABLE\_TEMPERATURE "Temperature"

#include <QWidget>

#include <QSqlDatabase>

#include <QFile>

#include <QDebug>

#include <QSqlQuery>

#include <QSqlQueryModel>

#include <cstring>

#include <fstream>

#include <iostream>

#include <sstream>

#include "database.h"

#include "addingnewweatherrecord.h"

#include "deleteweatherrecord.h"

#include "modifyweatherrecord.h"

#include "searchrecord.h"

#include "sorting.h"

namespace Ui {

class MainDataBase;

}

class MainDataBase : public QWidget

{

Q\_OBJECT

signals:

void showMainDB();

public:

int countOfEncryptedWords = 0;

std::string stringToDecrypt;

explicit MainDataBase(QWidget \*parent = nullptr);

virtual ~*MainDataBase*();

bool createMainTable();

void connectToDataBase();

void setupMainModel(const QStringList &headers);

bool insertIntoMainTable(const QVariantList &data);

bool mainDataBaseNotEmpty();

void refreshWindow();

void createUI();

bool openDataBase();

bool restoreDataBase();

void closeDataBase();

QVariantList EncryptionDecryption();

void decryptWeatherDB();

void xor\_encryption(const std::string &key, std::string &str);

QPushButton \*getAddNewButton();

QPushButton \*getEncodeButton();

QPushButton \*getDecodeButton();

QPushButton \*getDeleteButton();

QPushButton \*getModifyButton();

QPushButton \*getSearchButton();

QPushButton \*getUndoButton();

QPushButton \*getSortingButton();

QLabel \*getLabel();

private slots:

void on\_exitButton\_clicked();

void on\_usersDBpushButton\_clicked();

void on\_refreshButton\_clicked();

void on\_addNewButton\_clicked();

void on\_encodeButton\_clicked();

void on\_deleteButton\_clicked();

void on\_modifyButton\_clicked();

void on\_decodeButton\_clicked();

void on\_searchButton\_clicked();

void on\_undoPushButton\_clicked();

void on\_sortingPushButton\_clicked();

private:

Ui::MainDataBase \*ui;

QSqlDatabase db;

DataBase \*weatherDB;

QSqlQueryModel \*modelMain;

AddingNewWeatherRecord \*weather;

DeleteWeatherRecord \*deleteWeather;

ModifyWeatherRecord \*modifyRecord;

SearchRecord \*search;

Sorting \*sort;

const char\* key = "!#\*($U()RI{<#FR#asdgnyu56UH@E#kkI@23FVS-ENJOY";

};

#endif // MAINDATABASE\_H

**8.2 Код файла maindatabase.cpp:**

#include "maindatabase.h"

#include "ui\_maindatabase.h"

MainDataBase::MainDataBase(QWidget \*parent) :

QWidget(parent),

ui(new Ui::MainDataBase)

{

ui->setupUi(this);

weatherDB = new DataBase();

this->connectToDataBase();

this->setupMainModel(QStringList() << "ID" << "День недели" << "Дата" << "Месяц"

<< "Основные характеристики" << "Температура");

this->createUI();

this->closeDataBase();

weather = new AddingNewWeatherRecord();

connect(weather, &AddingNewWeatherRecord::showWindow, this, &MainDataBase::show);

deleteWeather = new DeleteWeatherRecord();

connect(deleteWeather, &DeleteWeatherRecord::showWindow, this, &MainDataBase::show);

modifyRecord = new ModifyWeatherRecord();

connect(modifyRecord, &ModifyWeatherRecord::showWindow, this, &MainDataBase::show);

ui->decodeButton->setEnabled(false);

ui->searchButton->hide();

ui->undoPushButton->hide();

ui->sortingPushButton->hide();

search = new SearchRecord();

connect(search, &SearchRecord::showWindow, this, &MainDataBase::show);

sort = new Sorting();

connect(sort, &Sorting::showWindow, this, &MainDataBase::show);

}

MainDataBase::~MainDataBase()

{

delete ui;

}

QPushButton \*MainDataBase::getAddNewButton(){

return ui->addNewButton;

}

QPushButton \*MainDataBase::getEncodeButton(){

return ui->encodeButton;

}

QPushButton \*MainDataBase::getDecodeButton(){

return ui->decodeButton;

}

QPushButton \*MainDataBase::getDeleteButton(){

return ui->deleteButton;

}

QPushButton \*MainDataBase::getModifyButton(){

return ui->modifyButton;

}

QPushButton \*MainDataBase::getSearchButton(){

return ui->searchButton;

}

QPushButton \*MainDataBase::getUndoButton(){

return ui->undoPushButton;

}

QLabel \*MainDataBase::getLabel(){

return ui->label;

}

QPushButton \*MainDataBase::getSortingButton(){

return ui->sortingPushButton;

}

void MainDataBase::connectToDataBase()

{

if(!QFile("/home/vladislav/Documents/StudyDocs/"

"3 semester/IT/Lab3/DataBaseOfUsers/DB/" DATABASE\_WEATHER\_NAME).exists()){

this->restoreDataBase();

} else {

this->openDataBase();

}

}

bool MainDataBase::restoreDataBase()

{

if(this->openDataBase()){

if((!this->createMainTable())){

return false;

} else {

return true;

}

} else {

return false;

}

return false;

}

bool MainDataBase::openDataBase()

{

this->closeDataBase();

db = QSqlDatabase::addDatabase("QSQLITE");

db.setHostName(DATABASE\_HOSTNAME\_WEATHER);

db.setDatabaseName("/home/vladislav/Documents/StudyDocs/3 semester/IT/"

"Lab3/DataBaseOfUsers/DB/" DATABASE\_WEATHER\_NAME);

if(db.open()){

return true;

} else {

return false;

}

}

void MainDataBase::closeDataBase()

{

db.close();

}

bool MainDataBase::createMainTable()

{

QSqlQuery query;

if(!query.exec("CREATE TABLE " TABLE\_WEATHER

"(ID INTEGER PRIMARY KEY AUTOINCREMENT, "

"Weekday DATE NOT NULL,"

"Date\_ DATE NOT NULL, "

"Month DATE NOT NULL, "

"Label TEXT NOT NULL, "

"Temperature TEXT NOT NULL)")){

return false;

} else {

return true;

}

return false;

}

void MainDataBase::on\_exitButton\_clicked()

{

this->close();

}

void MainDataBase::setupMainModel(const QStringList &headers)

{

modelMain = new QSqlQueryModel(this);

modelMain->setQuery("SELECT \* "

"FROM " TABLE\_WEATHER);

for(int i = 0, j = 0; i < modelMain->columnCount(); i++, j++){

modelMain->setHeaderData(i, Qt::Horizontal, headers[j]);

}

}

void MainDataBase::createUI()

{

ui->tableView->setModel(modelMain);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->resizeColumnsToContents();

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->tableView->horizontalHeader()->setStretchLastSection(true);

}

bool MainDataBase::insertIntoMainTable(const QVariantList &data){

QSqlQuery query;

query.prepare("INSERT INTO " TABLE\_WEATHER " ( " TABLE\_WEEKDAY ", "

TABLE\_DATE ", " TABLE\_MONTH ", "

TABLE\_LABEL ", " TABLE\_TEMPERATURE ")"

"VALUES (:Weekday, :Date\_, :Month, :Label, :Temperature)");

query.bindValue(":Weekday", data[0].toString());

query.bindValue(":Date\_", data[1].toString());

query.bindValue(":Month", data[2].toString());

query.bindValue(":Label", data[3].toString());

query.bindValue(":Temperature", data[4].toString());

if(!query.exec()){

qDebug() << "error insert into " << TABLE\_WEATHER;

return false;

} else {

return true;

}

return false;

}

void MainDataBase::refreshWindow(){

this->close();

this->connectToDataBase();

this->setupMainModel(QStringList() << "ID" << "День недели" << "Дата" << "Месяц"

<< "Основные характеристики" << "Температура");

this->createUI();

this->show();

}

bool MainDataBase::mainDataBaseNotEmpty(){

this->connectToDataBase();

QSqlQuery query;

QString queryString = QString("SELECT COUNT(\*) "

"FROM " TABLE\_WEATHER);

query.prepare(queryString);

query.exec();

query.next();

if(query.value(0).toInt() > 0){

return true;

}

else return false;

}

void MainDataBase::on\_refreshButton\_clicked()

{

this->refreshWindow();

}

void MainDataBase::on\_addNewButton\_clicked()

{

weather->show();

}

void MainDataBase::xor\_encryption(const std::string &key, std::string &str){

for(size\_t i = 0; i != str.length(); i++)

str[i] ^= key[i % key.size()];

}

QVariantList MainDataBase::EncryptionDecryption(){

QVariantList encryptedData;

DataBaseReadMethods \*readFromDB = new DataBaseReadMethods();

QVariantList data = readFromDB->readDataFromWeatherSqlFileToQVariantList();

std::string str;

int countOfSpaces = 0;

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

str.append(data.value(i).toString().toStdString());

str.append(" ");

}

for(size\_t i = 0; i < str.length(); i++){

if(str[i] == ' '){

countOfSpaces++;

if(countOfSpaces % 5 == 0){

str[i] = '\n';

if(str[i - 1] == ' ')

str.erase(i - 1, 1);

}

}

}

stringToDecrypt = str;

std::stringstream strStream(str);

std::string word;

QVariant wordQVariant;

weatherDB->deleteAllWeatherRecords();

weatherDB->createWeatherTable();

countOfEncryptedWords = countOfSpaces;

for(int i = 1; i <= countOfEncryptedWords; i++){

strStream >> word;

this->xor\_encryption(key, word);

wordQVariant = QString::fromStdString(word);

encryptedData.append(wordQVariant);

if(i % 5 == 0){

weatherDB->insertIntoWeatherTable(encryptedData);

encryptedData.clear();

}

}

QMessageBox msgBox;

msgBox.setWindowTitle("Шифрование");

msgBox.setText("Информация успешно зашифрована!");

msgBox.exec();

ui->addNewButton->setEnabled(false);

ui->deleteButton->setEnabled(false);

ui->modifyButton->setEnabled(false);

ui->encodeButton->setEnabled(false);

ui->decodeButton->setEnabled(true);

this->refreshWindow();

delete readFromDB;

return encryptedData;

}

void MainDataBase::on\_encodeButton\_clicked()

{

this->EncryptionDecryption();

this->refreshWindow();

}

void MainDataBase::on\_deleteButton\_clicked()

{

deleteWeather->show();

}

void MainDataBase::on\_modifyButton\_clicked()

{

modifyRecord->show();

}

void MainDataBase::decryptWeatherDB(){

std::stringstream strStream(stringToDecrypt);

DataBaseReadMethods \*readFromDB = new DataBaseReadMethods();

QVariantList encryptedData = readFromDB->readDataFromWeatherSqlFileToQVariantList();

QVariantList decryptedData;

QString qstr;

QVariant qvar;

std::string str;

weatherDB->deleteAllWeatherRecords();

weatherDB->createWeatherTable();

for(int i = 1; i <= countOfEncryptedWords; i++){

strStream >> stringToDecrypt;

str = stringToDecrypt;

this->xor\_encryption(key, str);

this->xor\_encryption(key, str);

qstr = QString::fromStdString(str);

qvar = qstr;

qvar = QString::fromStdString(str);

decryptedData.append(qvar);

if(i % 5 == 0){

weatherDB->insertIntoWeatherTable(decryptedData);

decryptedData.clear();

}

}

QMessageBox msgBox;

msgBox.setWindowTitle("Дешифрование");

msgBox.setText("Информация успешно дешифрована!");

msgBox.exec();

ui->addNewButton->setEnabled(true);

ui->deleteButton->setEnabled(true);

ui->modifyButton->setEnabled(true);

ui->encodeButton->setEnabled(true);

ui->decodeButton->setEnabled(false);

delete readFromDB;

}

void MainDataBase::on\_decodeButton\_clicked()

{

this->decryptWeatherDB();

this->refreshWindow();

}

void MainDataBase::on\_searchButton\_clicked()

{

search->show();

ui->undoPushButton->setEnabled(true);

}

void MainDataBase::on\_undoPushButton\_clicked()

{

weatherDB->undoMethod();

}

void MainDataBase::on\_sortingPushButton\_clicked()

{

sort->show();

ui->undoPushButton->setEnabled(true);

}

**9.1 Код файла main.cpp:**

#include <QApplication>

#include <QMessageBox>

#include "mainwindow.h"

#include "htmlThief.h"

#include "htmlParser.h"

int main(int argc, char \*argv[])

{

Sinoptik sinoptik;

QApplication a(*argc*, argv);

MainWindow w;

w.hide();

QMessageBox msgBox;

msgBox.setWindowTitle("Копирование");

msgBox.setText("Успешно скопирована информация с сайта sinoptik.com");

msgBox.show();

system(qPrintable("./htmlThief https://sinoptik.com.ru/%D0%BF%D0%BE%D0%B3%D0%BE%D0%B4%D0%B0-%D1%81%D0%B0%D0%BD%D0%BA%D 1%82-%D0%BF%D0%B5%D1%82%D0%B5%D1%80%D0%B1%D1%83%D1%80%D0%B3"));

sinoptik.htmlParse();

sinoptik.fillWeatherDB();

return a.exec();

}

**10.1 Код файла mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QWidget>

#include <QMainWindow>

#include <QSqlQueryModel>

#include "database.h"

#include "loginning.h"

#include "maindatabase.h"

#include "usersediditing.h"

#include "deleteuserfromdb.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QWidget

{

Q\_OBJECT

signals:

void showMainWindow();

public:

explicit MainWindow(QWidget \*parent = 0);

virtual ~*MainWindow*();

private slots:

void on\_exitPushButton\_clicked();

void on\_addNewUserPushButton\_clicked();

void on\_refreshPushButton\_clicked();

void on\_deletePushButton\_clicked();

private:

Ui::MainWindow \*ui;

loginning \*loginWindow;

UsersEdiditing \*userEdit;

deleteUserFromDB \*deleteUser;

DataBase \*db;

QSqlQueryModel \*modelMain;

public:

void refreshWindow();

private:

void setupMainModel(const QStringList &headers);

void createUI();

};

#endif // MAINWINDOW\_H

**10.2 Код файла mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <iostream>

MainWindow::MainWindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

db = new DataBase();

if(db->checkingForRecord("admin")){

QVariantList data;

data.append("admin");

data.append("admin");

db->insertIntoMainTable(data);

data.clear();

}

this->setupMainModel(QStringList() << "ID" << "Логин" << "Пароль");

this->createUI();

loginWindow = new loginning();

connect(loginWindow, &loginning::showWindow, this, &MainWindow::show);

loginWindow->show();

userEdit = new UsersEdiditing();

connect(userEdit, &UsersEdiditing::showWindow, this, &MainWindow::show);

deleteUser = new deleteUserFromDB();

connect(deleteUser, &deleteUserFromDB::showWindow, this, &MainWindow::show);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::setupMainModel(const QStringList &headers)

{

modelMain = new QSqlQueryModel(this);

modelMain->setQuery("SELECT \* FROM " TABLE);

for(int i = 0, j = 0; i < modelMain->columnCount(); i++, j++){

modelMain->setHeaderData(i, Qt::Horizontal, headers[j]);

}

}

void MainWindow::createUI()

{

ui->tableView->setModel(modelMain);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->resizeColumnsToContents();

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->tableView->horizontalHeader()->setStretchLastSection(true);

}

void MainWindow::on\_exitPushButton\_clicked()

{

this->close();

}

void MainWindow::on\_addNewUserPushButton\_clicked()

{

userEdit->show();

}

void MainWindow::refreshWindow(){

this->close();

this->setupMainModel(QStringList() << "ID" << "Логин" << "Пароль");

this->createUI();

this->show();

}

void MainWindow::on\_refreshPushButton\_clicked()

{

this->refreshWindow();

}

void MainWindow::on\_deletePushButton\_clicked()

{

deleteUser->show();

}

**11.1 Код файла modifyweatherrecord.h:**

#ifndef MODIFYWEATHERRECORD\_H

#define MODIFYWEATHERRECORD\_H

#include <QWidget>

#include <QVariantList>

#include <QMessageBox>

#include "database.h"

namespace Ui {

class ModifyWeatherRecord;

}

class ModifyWeatherRecord : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit ModifyWeatherRecord(QWidget \*parent = nullptr);

~*ModifyWeatherRecord*();

private slots:

void on\_pushButton\_clicked();

private:

Ui::ModifyWeatherRecord \*ui;

DataBase \*db;

};

#endif // MODIFYWEATHERRECORD\_H

**11.2 Код файла modifyweatherrecord.cpp:**

#include "modifyweatherrecord.h"

#include "ui\_modifyweatherrecord.h"

ModifyWeatherRecord::ModifyWeatherRecord(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ModifyWeatherRecord)

{

ui->setupUi(this);

db = new DataBase();

}

ModifyWeatherRecord::~*ModifyWeatherRecord*()

{

delete ui;

}

void ModifyWeatherRecord::on\_pushButton\_clicked()

{

QVariantList data;

QString weekday = ui->weekdayLineEdit->text();

QString date = ui->dateLineEdit->text();

QString month = ui->monthLineEdit->text();

QString label = ui->labelLineEdit->text();

QString temperature = ui->temperatureLineEdit->text() + "°";

int id = ui->IDLineEdit->text().toInt();

for(int i = 0; i < weekday.length(); i++){

if(weekday[i] == ' ')

weekday[i] = '\_';

}

for(int i = 0; i < date.length(); i++){

if(date[i] == ' ')

date[i] = '\_';

}

for(int i = 0; i < month.length(); i++){

if(month[i] == ' ')

month[i] = '\_';

}

for(int i = 0; i < label.length(); i++){

if(label[i] == ' ')

label[i] = '\_';

}

for(int i = 0; i < temperature.length(); i++){

if(temperature[i] == ' ')

temperature[i] = '\_';

}

data.append(weekday);

data.append(date);

data.append(month);

data.append(label);

data.append(temperature);

if(db->checkingForIDInWeatherDB(id)){

if(weekday == nullptr || date == nullptr || month == nullptr ||

label == nullptr || temperature == nullptr || ui->IDLineEdit->text() == nullptr){

QMessageBox::warning(this, "Внимание", "Некоторые поля не заполнены.");

}

else if(weekday != nullptr && date != nullptr && month != nullptr &&

label != nullptr && temperature != nullptr && ui->IDLineEdit->text() != nullptr){

db->modifyWeatherRecordWithID(data, id);

QMessageBox msgBox;

msgBox.setWindowTitle("Изменение");

msgBox.setText("Запись была успешно изменена.");

msgBox.*exec*();

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

}

else if(!db->checkingForIDInWeatherDB(id)){

QMessageBox::warning(this, "Внимание", "Вы ввели неверный ID.");

}

else{

QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

ui->IDLineEdit->clear();

ui->weekdayLineEdit->clear();

ui->dateLineEdit->clear();

ui->monthLineEdit->clear();

ui->labelLineEdit->clear();

ui->temperatureLineEdit->clear();

data.clear();

this->close();

}

**12.1 Код файла searchrecord.h:**

#ifndef SEARCHRECORD\_H

#define SEARCHRECORD\_H

#include <QWidget>

#include <QMessageBox>

#include "database.h"

namespace Ui {

class SearchRecord;

}

class SearchRecord : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit SearchRecord(QWidget \*parent = nullptr);

~SearchRecord();

private slots:

void on\_pushButton\_clicked();

void on\_IDradioButton\_clicked();

void on\_weekdayRadioButton\_clicked();

void on\_dateRadioButton\_clicked();

void on\_monthRadioButton\_clicked();

void on\_genCharacRadioButton\_clicked();

void on\_temperatureRadioButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::SearchRecord \*ui;

DataBase \*db;

};

#endif // SEARCHRECORD\_H

**12.2 Код файла searchrecord.cpp:**

#include "searchrecord.h"

#include "ui\_searchrecord.h"

SearchRecord::SearchRecord(QWidget \*parent) :

QWidget(parent),

ui(new Ui::SearchRecord)

{

ui->setupUi(this);

db = new DataBase();

}

SearchRecord::~SearchRecord()

{

delete ui;

}

void SearchRecord::on\_pushButton\_clicked()

{

if(ui->IDLineEdit->text() != nullptr && ui->IDradioButton->isChecked() == true){

if(db->checkingForIDInWeatherDB(ui->IDLineEdit->text().toInt())){

db->searchByID(ui->IDLineEdit->text().toInt());

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по ID");

msgBox.setText("Поиск по ID произведён.");

msgBox.exec();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по ID");

msgBox.setText("Не найдено записи с таким ID.");

msgBox.exec();

}

}

else if(ui->IDradioButton->isChecked() == true)

QMessageBox::warning(this, "Внимание", "Поле ID пустое!");

if(ui->weekdayLineEdit->text() != nullptr && ui->weekdayRadioButton->isChecked() == true){

if(!db->checkingForWeekdayInWeatherDB(ui->weekdayLineEdit->text())){

db->searchByWeekday(ui->weekdayLineEdit->text());

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по дню недели");

msgBox.setText("Поиск по дню недели произведён.");

msgBox.exec();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по дню недели");

msgBox.setText("Не найдено записи с таким днем недели.");

msgBox.exec();

}

}

else if(ui->weekdayRadioButton->isChecked() == true)

QMessageBox::warning(this, "Внимание", "Поле День Недели пустое!");

if(ui->dateLineEdit->text() != nullptr && ui->dateRadioButton->isChecked() == true){

db->searchByDate(ui->dateLineEdit->text());

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по дате");

msgBox.setText("Поиск по дате произведён.");

msgBox.exec();

}

else if(ui->dateRadioButton->isChecked() == true)

QMessageBox::warning(this, "Внимание", "Поле Дата пустое!");

if(ui->monthLineEdit->text() != nullptr && ui->monthRadioButton->isChecked() == true){

db->searchByMonth(ui->monthLineEdit->text());

QMessageBox msgBox;

msgBox.setWindowTitle("Поиск по месяцу");

msgBox.setText("Поиск по месяцу произведён.");

msgBox.exec();

}

else if(ui->monthRadioButton->isChecked() == true)

QMessageBox::warning(this, "Внимание", "Поле Месяц пустое!");

ui->IDLineEdit->clear();

ui->dateLineEdit->clear();

ui->monthLineEdit->clear();

ui->weekdayLineEdit->clear();

this->close();

}

void SearchRecord::on\_IDradioButton\_clicked()

{

ui->IDLineEdit->setEnabled(true);

ui->weekdayRadioButton->setEnabled(false);

ui->dateRadioButton->setEnabled(false);

ui->monthRadioButton->setEnabled(false);

}

void SearchRecord::on\_weekdayRadioButton\_clicked()

{

ui->weekdayLineEdit->setEnabled(true);

ui->IDradioButton->setEnabled(false);

ui->dateRadioButton->setEnabled(false);

ui->monthRadioButton->setEnabled(false);

}

void SearchRecord::on\_dateRadioButton\_clicked()

{

ui->dateLineEdit->setEnabled(true);

ui->weekdayRadioButton->setEnabled(false);

ui->IDradioButton->setEnabled(false);

ui->monthRadioButton->setEnabled(false);

}

void SearchRecord::on\_monthRadioButton\_clicked()

{

ui->monthLineEdit->setEnabled(true);

ui->weekdayRadioButton->setEnabled(false);

ui->dateRadioButton->setEnabled(false);

ui->IDradioButton->setEnabled(false);

}

void SearchRecord::on\_pushButton\_2\_clicked()

{

ui->IDradioButton->setDisabled(true);

ui->weekdayRadioButton->setDisabled(true);

ui->dateRadioButton->setDisabled(true);

ui->monthRadioButton->setDisabled(true);

ui->IDradioButton->setEnabled(true);

ui->weekdayRadioButton->setEnabled(true);

ui->dateRadioButton->setEnabled(true);

ui->monthRadioButton->setEnabled(true);

ui->IDLineEdit->setEnabled(false);

ui->weekdayLineEdit->setEnabled(false);

ui->dateLineEdit->setEnabled(false);

ui->monthLineEdit->setEnabled(false);

}

**13.1 Код файла sorting.h:**

#ifndef SORTING\_H

#define SORTING\_H

#include <QWidget>

#include "database.h"

namespace Ui {

class Sorting;

}

class Sorting : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit Sorting(QWidget \*parent = nullptr);

~*Sorting*();

private slots:

void on\_sortByWeekdayButton\_clicked();

void on\_sortByDateButton\_clicked();

void on\_sortByMonth\_clicked();

void on\_sortByGenChButton\_clicked();

void on\_sortByTemperature\_clicked();

private:

Ui::Sorting \*ui;

SortMethods \*sort;

};

#endif // SORTING\_H

**13.2 Код файла sorting.cpp:**

#include "sorting.h"

#include "ui\_sorting.h"

Sorting::Sorting(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Sorting)

{

ui->setupUi(this);

sort = new SortMethods();

}

Sorting::~Sorting()

{

delete ui;

}

void Sorting::on\_sortByWeekdayButton\_clicked()

{

sort->sortingByWeekday();

this->close();

}

void Sorting::on\_sortByDateButton\_clicked()

{

sort->sortingByDate();

this->close();

}

void Sorting::on\_sortByMonth\_clicked()

{

sort->sortingByMonth();

this->close();

}

void Sorting::on\_sortByGenChButton\_clicked()

{

sort->sortingByGeneralCharacteristics();

this->close();

}

void Sorting::on\_sortByTemperature\_clicked()

{

sort->sortingByTemperature();

this->close();

}

**14.1 Код файла userediting.h:**

#ifndef USERSEDIDITING\_H

#define USERSEDIDITING\_H

#include <QWidget>

#include <QMessageBox>

#include <cstring>

#include "database.h"

namespace Ui {

class UsersEdiditing;

}

class UsersEdiditing : public QWidget

{

Q\_OBJECT

signals:

void showWindow();

public:

explicit UsersEdiditing(QWidget \*parent = nullptr);

~*UsersEdiditing*();

private slots:

void on\_addNewUserPushButton\_clicked();

private:

Ui::UsersEdiditing \*ui;

DataBase \*db;

};

#endif // USERSEDIDITING\_H

**14.2 Код файла userediting.cpp:**

#include "usersediditing.h"

#include "ui\_usersediditing.h"

UsersEdiditing::UsersEdiditing(QWidget \*parent) :

QWidget(parent),

ui(new Ui::UsersEdiditing)

{

ui->setupUi(this);

db = new DataBase();

}

UsersEdiditing::~UsersEdiditing()

{

delete ui;

}

void UsersEdiditing::on\_addNewUserPushButton\_clicked()

{

if(db->checkingForIDInWeatherDB(ui->whichLine->text().toInt())){

if(ui->loginLineEdit->text() != nullptr && ui->psswdLineEdit->text() != nullptr &&

ui->whichLine->text() != nullptr){

if(ui->loginLineEdit->text() != "admin"){

if(db->checkingForRecord(ui->loginLineEdit->text())){

QString login = ui->loginLineEdit->text();

QString psswd = ui->psswdLineEdit->text();

int id = ui->whichLine->text().toInt();

QVariantList data;

data.append(login);

data.append(psswd);

if(db->checkingForID(id)){

db->insertIntoMainTableWithKnownID(data, id);

QMessageBox \_msgBox;

\_msgBox.setWindowTitle("Добавление");

\_msgBox.setText("Пользователь успешно добавлен.");

\_msgBox.exec();

}

else if(!db->checkingForID(id)){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поле с таким ID уже занято.");

msgBox.exec();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Произошла неизвестная ошибка!");

msgBox.exec();

}

data.clear();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

this->close();

}

else if(!db->checkingForRecord(ui->loginLineEdit->text())){

QMessageBox msgBox;

msgBox.setWindowTitle("Warning");

msgBox.setText("Can't add new user with this login.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Произошла неизвестная ошибка!");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

}

else if(ui->loginLineEdit->text() == "admin"){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Невозможно добавиь пользователя с логином \"admin\"");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Произошла неизвестная ошибка!");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

}

else if(ui->psswdLineEdit->text() != nullptr && ui->whichLine->text() != nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поле с логином не заполнено.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else if(ui->loginLineEdit->text() != nullptr && ui->psswdLineEdit->text() != nullptr){

QString login = ui->loginLineEdit->text();

QString psswd = ui->psswdLineEdit->text();

QVariantList data;

data.append(login);

data.append(psswd);

db->insertIntoMainTable(data);

data.clear();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

QMessageBox \_msgBox;

\_msgBox.setWindowTitle("Добавление");

\_msgBox.setText("Пользователь успешно добавлен.");

\_msgBox.exec();

this->close();

}

else if(ui->loginLineEdit->text() != nullptr && ui->whichLine->text() != nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поле пароль не заполнено.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else if(ui->loginLineEdit->text() != nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поля пароль и ID не заполнены.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else if(ui->psswdLineEdit->text() != nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поля логин и ID не заполнены.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else if(ui->whichLine->text() != nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Поля логин и пароль не заполнены.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else if(ui->loginLineEdit->text() == nullptr && ui->psswdLineEdit->text() == nullptr &&

ui->whichLine->text() == nullptr){

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Все поля пустые.");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

else{

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Произошла неизвестная ошибка!");

msgBox.exec();

ui->loginLineEdit->clear();

ui->psswdLineEdit->clear();

ui->whichLine->clear();

}

}

else if(!db->checkingForIDInWeatherDB(ui->whichLine->text().toInt())){

QMessageBox::warning(this, "Внимание", "ID не может быть меньше или равно 0.");

}

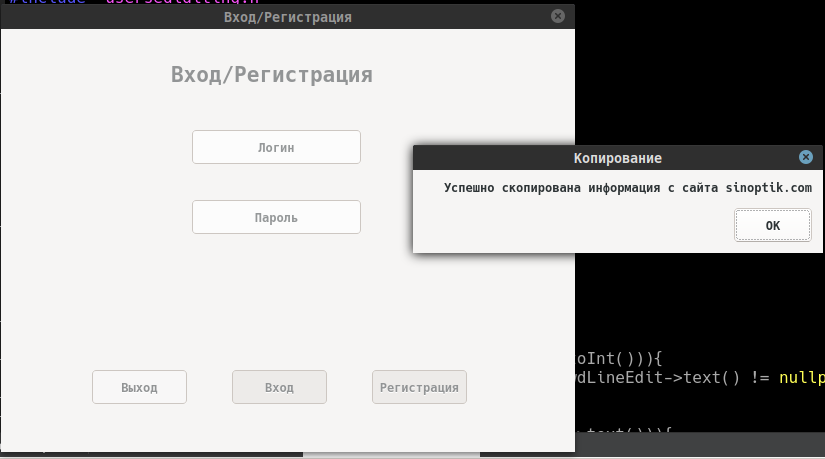
else{

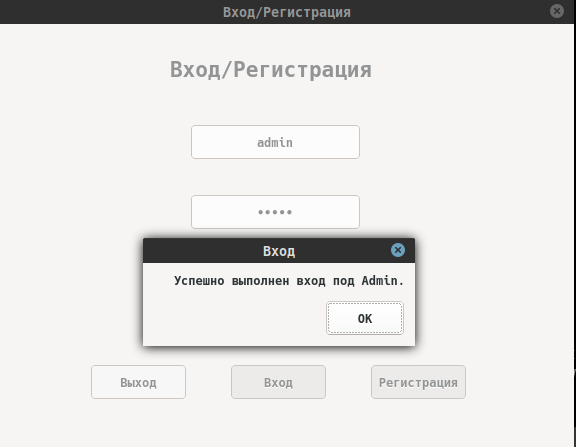
QMessageBox::warning(this, "Внимание", "Произошла неизвестная ошибка!");

}

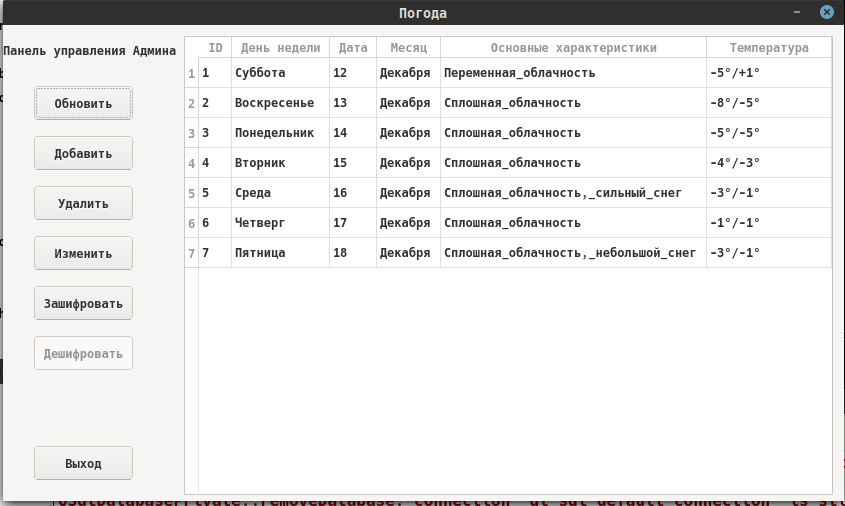
}

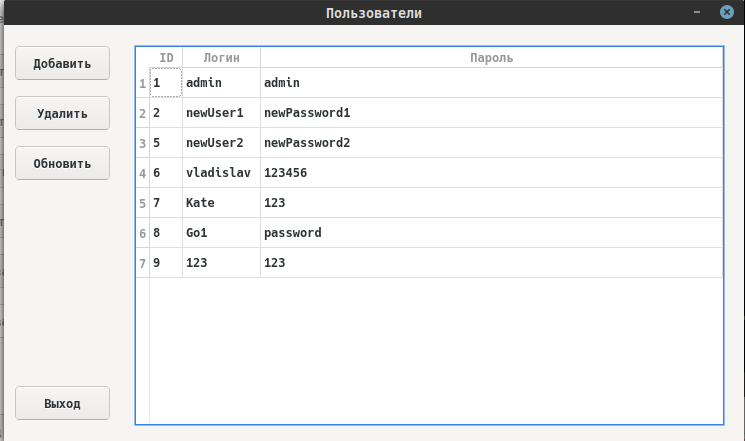
**Демонстрация работоспособности программы**

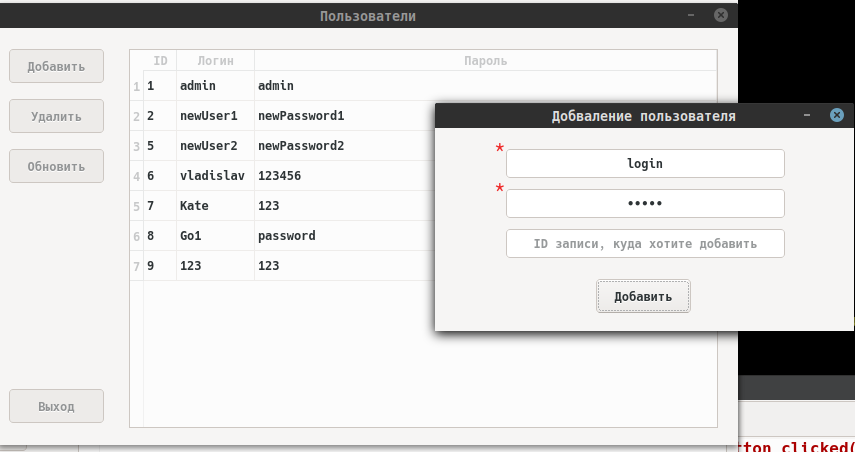
**Рис. 2 –** “Запуск программы, параллельно происходит копирование данных с сайта”

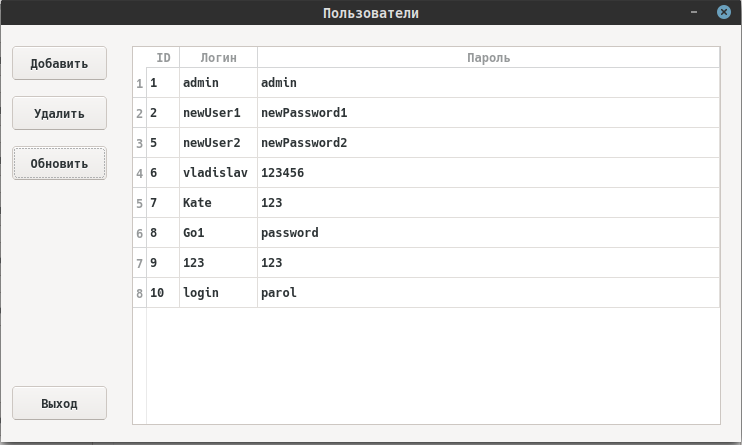
****

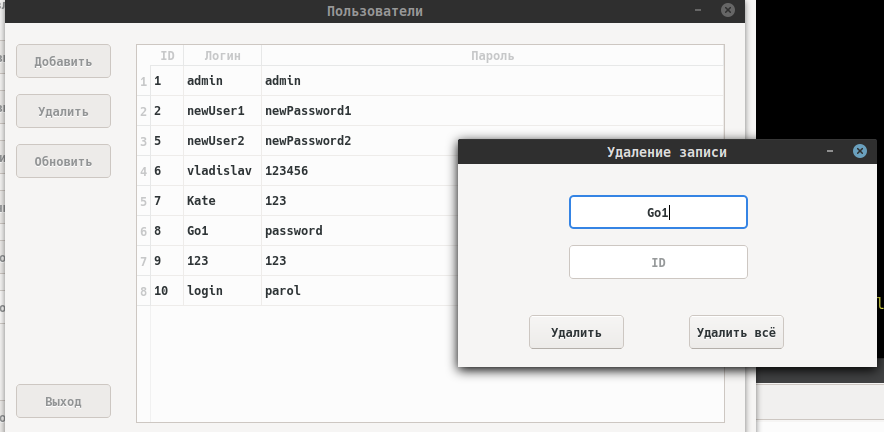
**Рис. 3 –** “Вход под привилегилированным пользователем”

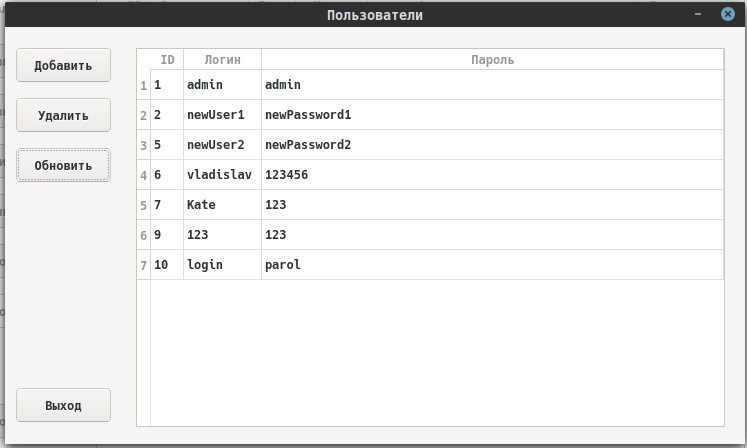
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 4 –** “Просмотр базы данных с погодой”

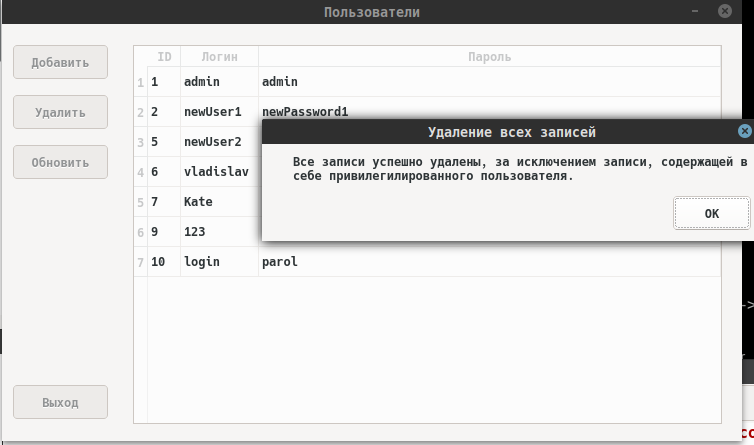
**Рис. 5 –** “Просмотр базы данных с пользователями”

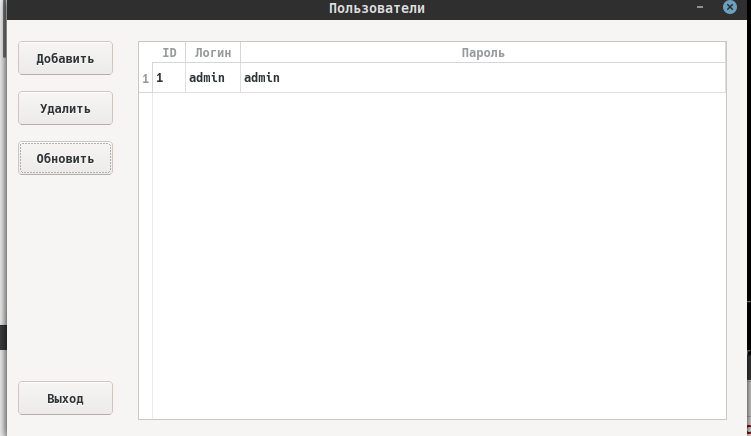
**Рис. 6 –** “Добавление нового пользователя посредством использования инструментов админа”

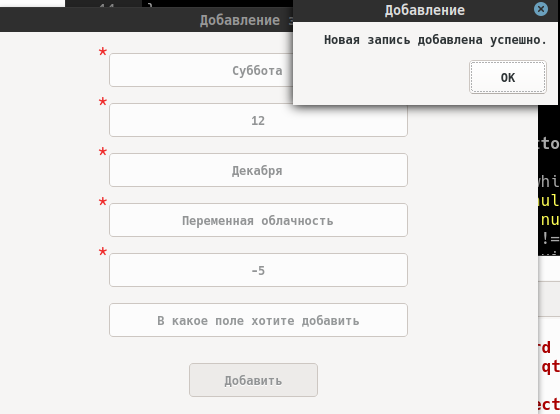
**Рис. 7 –** “Просмотр базы данных с пользователями после добавления”

**Рис. 8 –** “Удаление через логин нового пользователя посредством использования инструментов админа”

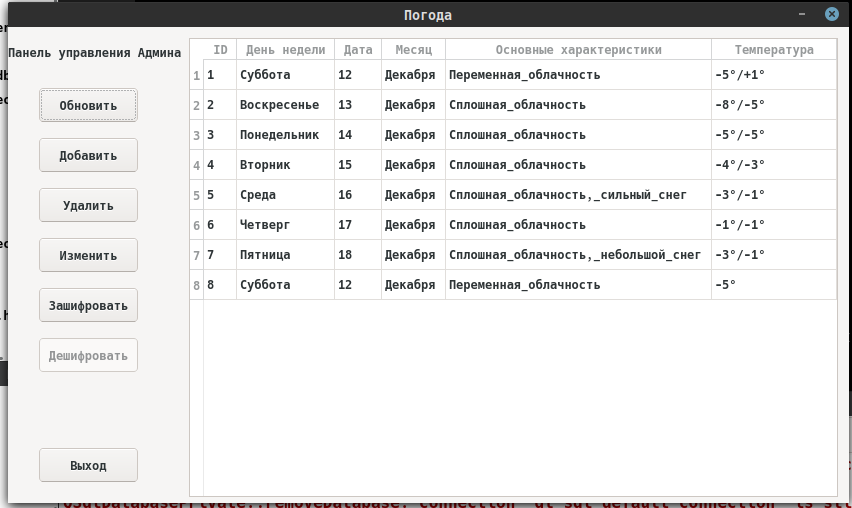
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 9 –** “Просмотр базы данных с пользователями после удаления”

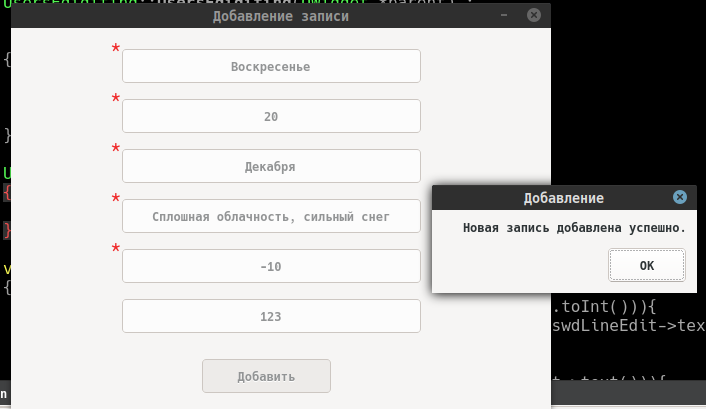
**Рис. 10 –** “Удаление всех записей из базы данных с пользователем, за исключением привилегилированного пользователя”

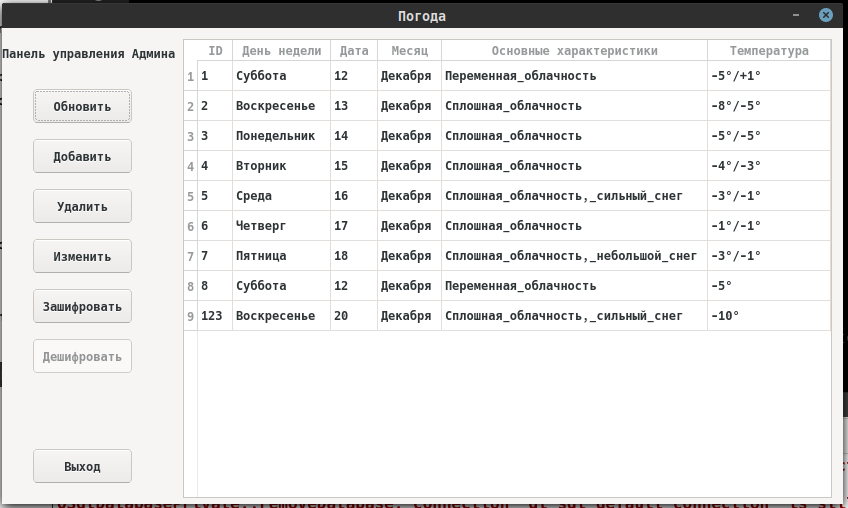
**Рис. 11 –** “Просмотр базы данных после удаления всех пользователей, за исключением привилегилированного пользователя”

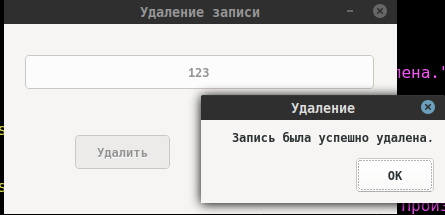
****

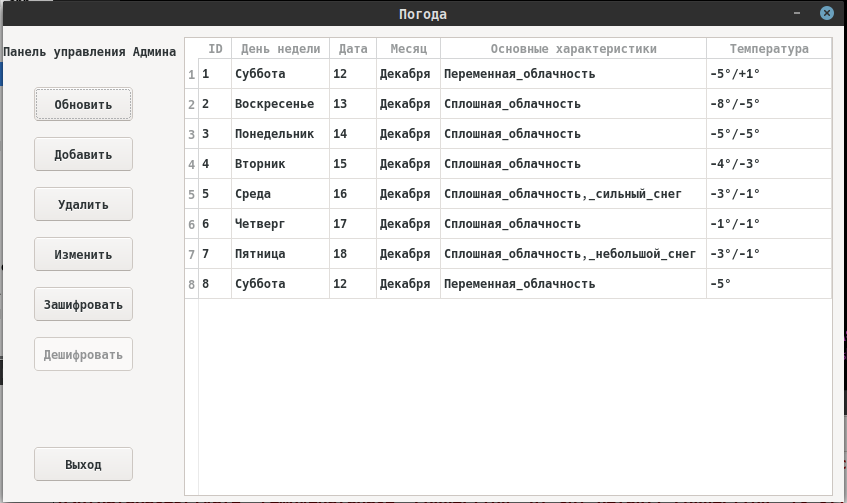
**Рис. 12 –** “Добавление новой записи в базу данных с погодой”

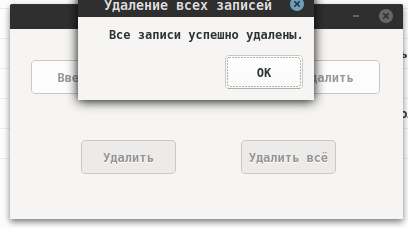
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 13 –** “Просмотр базы данных после добавления записи”

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 14 –** “Добавление новой записи в определенную позицию в базу данных с погодой”

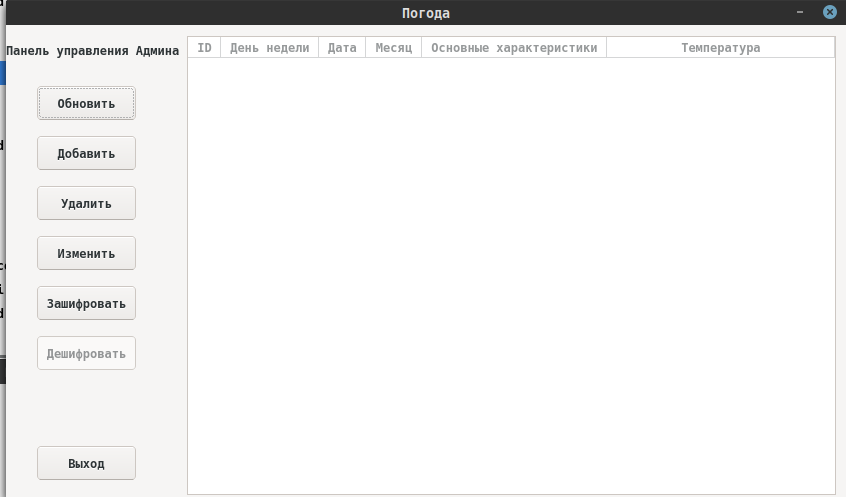
**Рис. 15 –** “Просмотр базы данных после добавления записи в определенную строку”

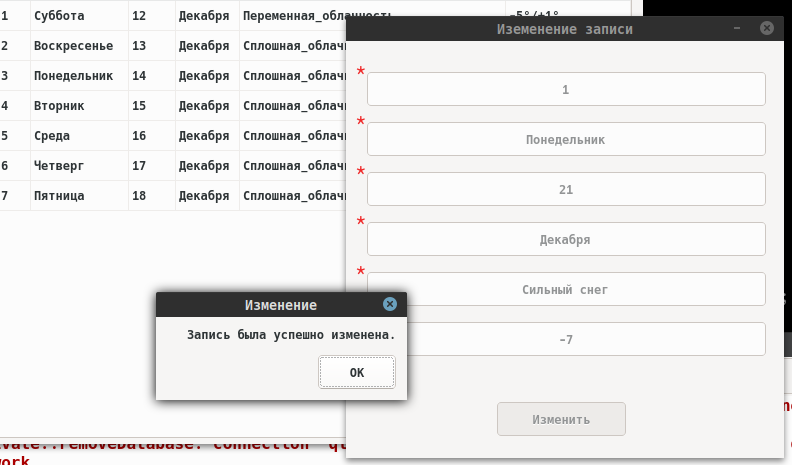
  
  
  
  
  
  
  
  
  
  
  
**Рис. 16 –** “Удаление записи по ID”

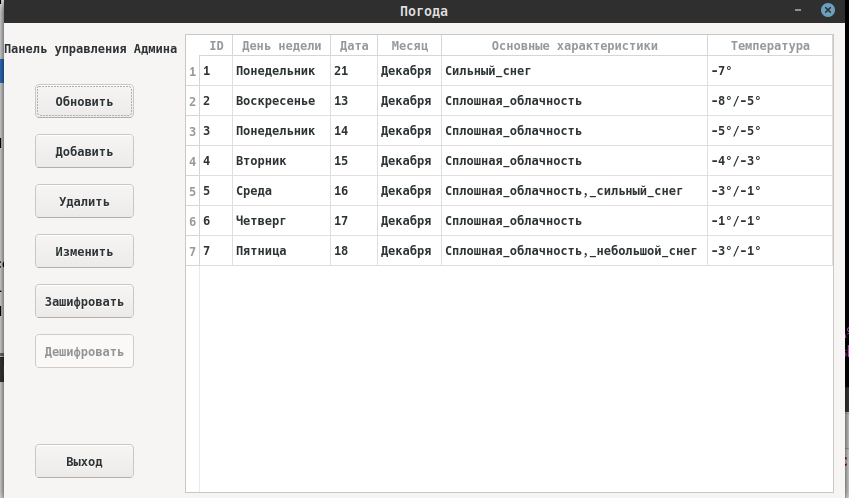
**Рис. 17 –** “Просмотр базы данных после удаления записи”

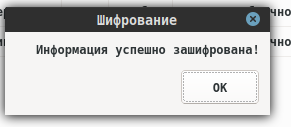


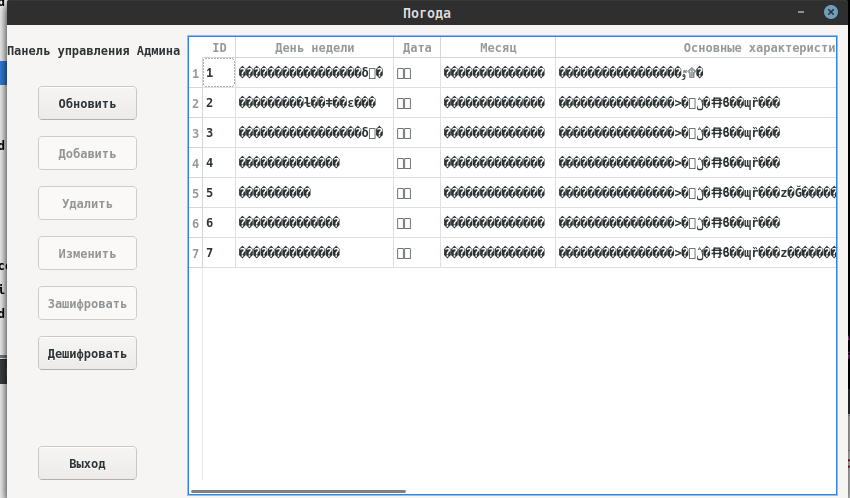
**Рис. 18 –** “Удаление всех записей из базы данных с погодой”

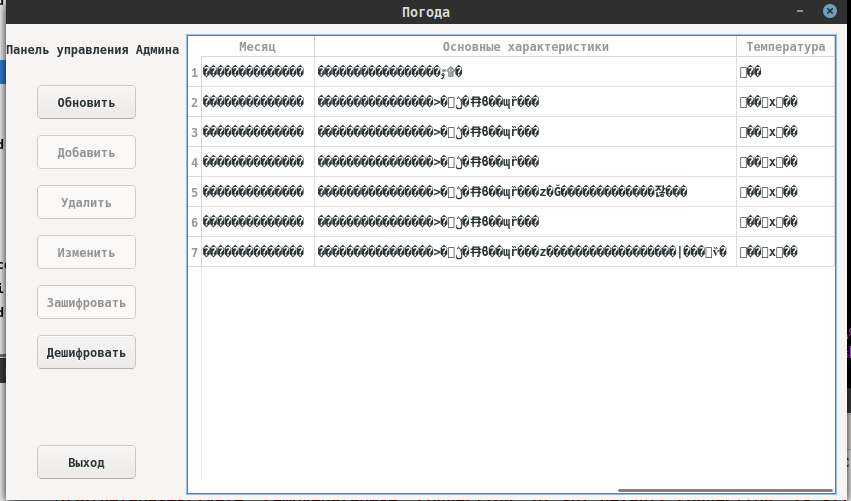
**Рис. 19 –** “Просмотр базы данных после удаления всех записей”

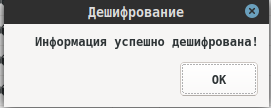
**Рис. 20 –** “Модификация записи в базе данных с погодой”

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 21 –** “Просмотр базы данных после модификации записи с ID = 1”

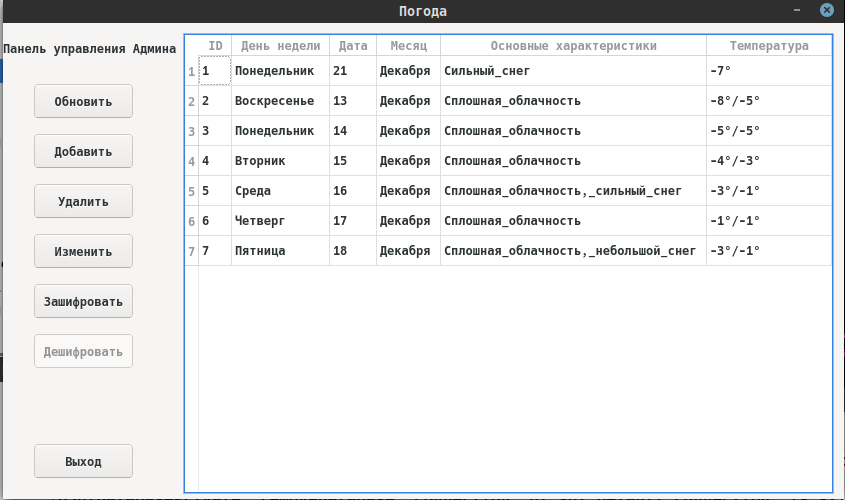
  
  
  
  
  
  
**Рис. 22 –** “Шифрование записей в базе данных”

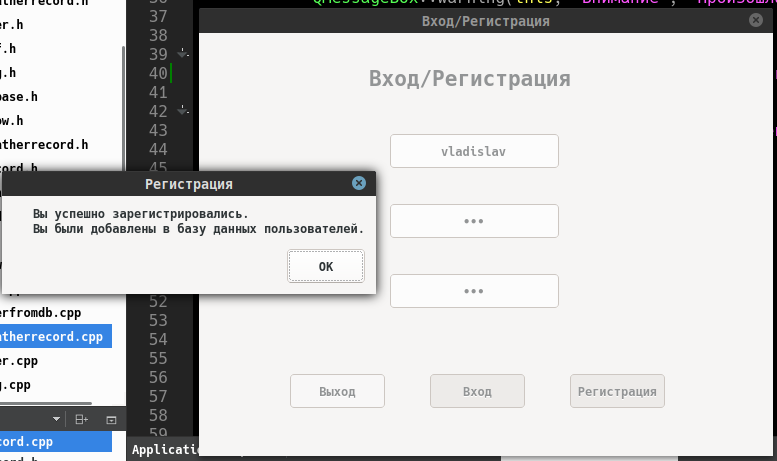
**Рис. 23 –** “Просмотр базы данных после шифрования записей, первые 4 колонки”

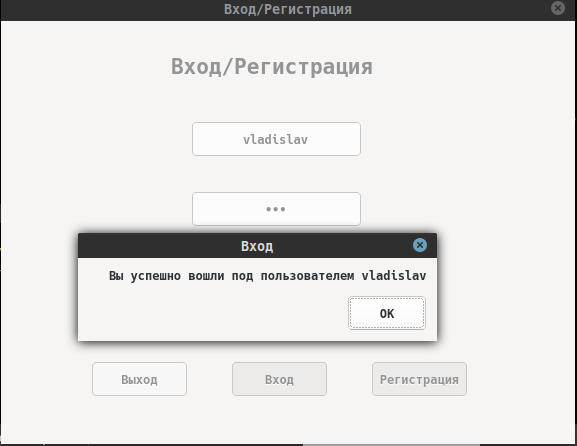
**Рис. 24 –** “Просмотр базы данных после шифрования остальные колонки”

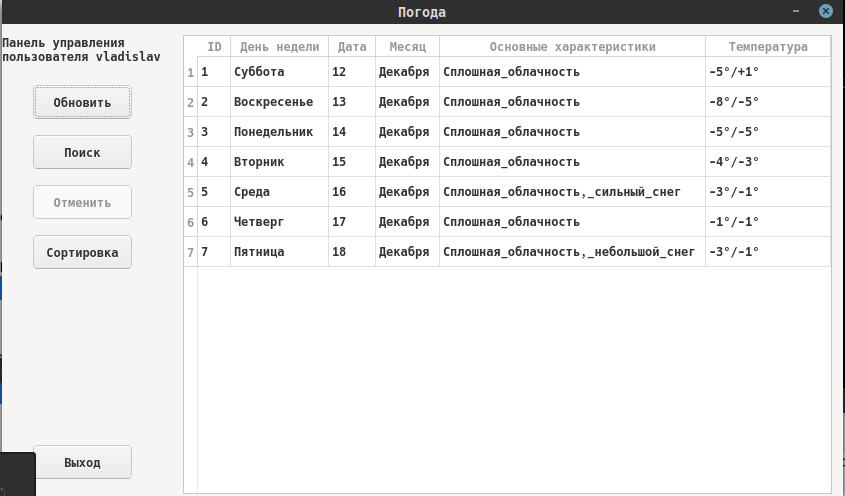


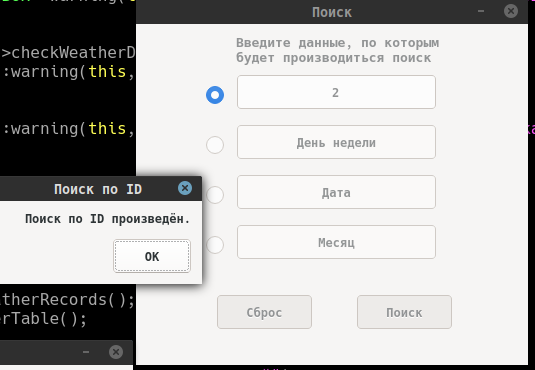
**Рис. 25 –** “Дешифрование записей

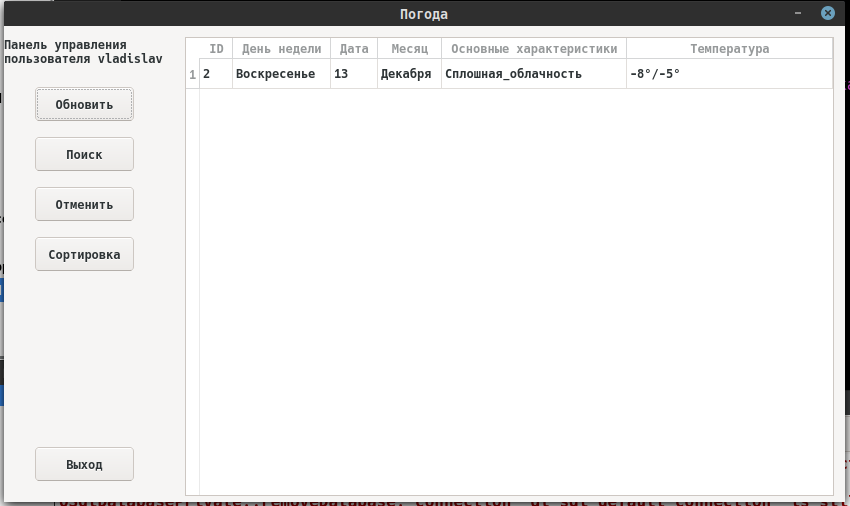
**Рис. 26 –** “Просмотр базы данных после дешифрования”

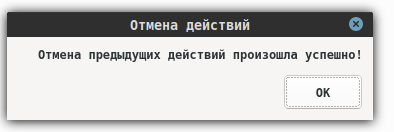
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 27 –** “Регистрация нового пользователя”

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 28 –** “Вход под новосозданным пользователем”

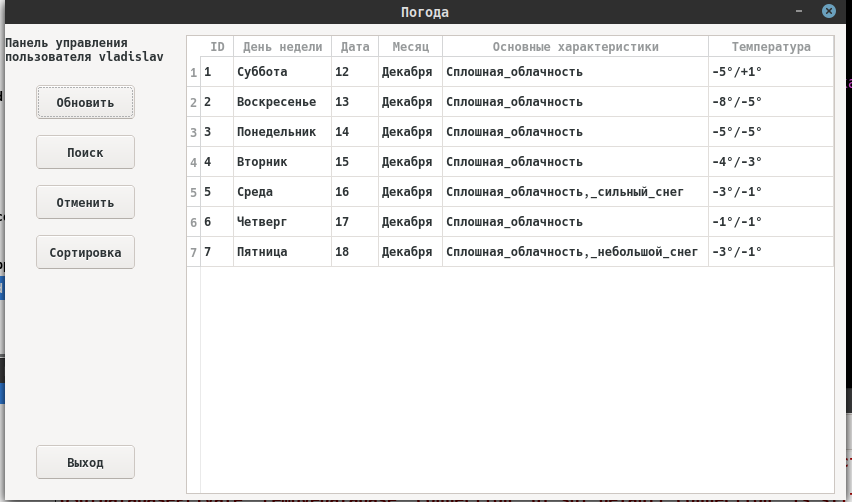
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 29 –** “Просмотр базы данных с панели управления пользователя”

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 30 –** “Поиск по ID”

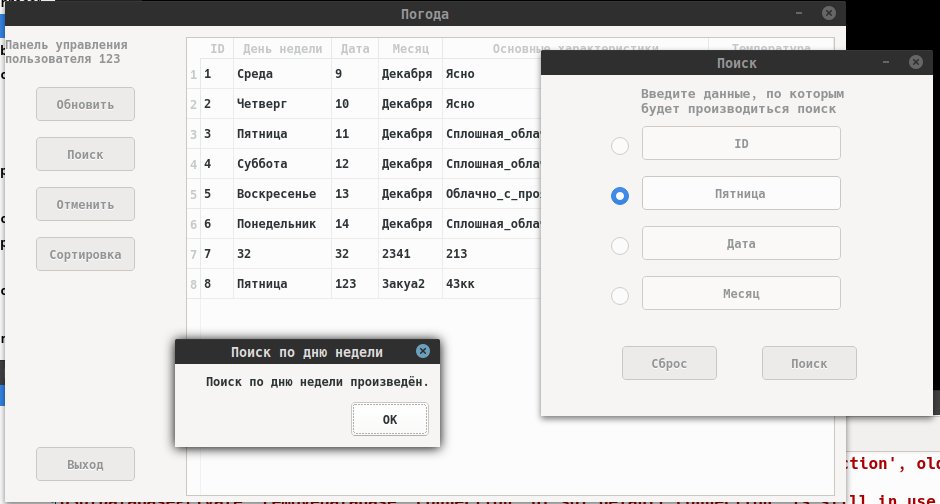
**Рис. 31 –** “Просмотр базы данных после поиска по ID”

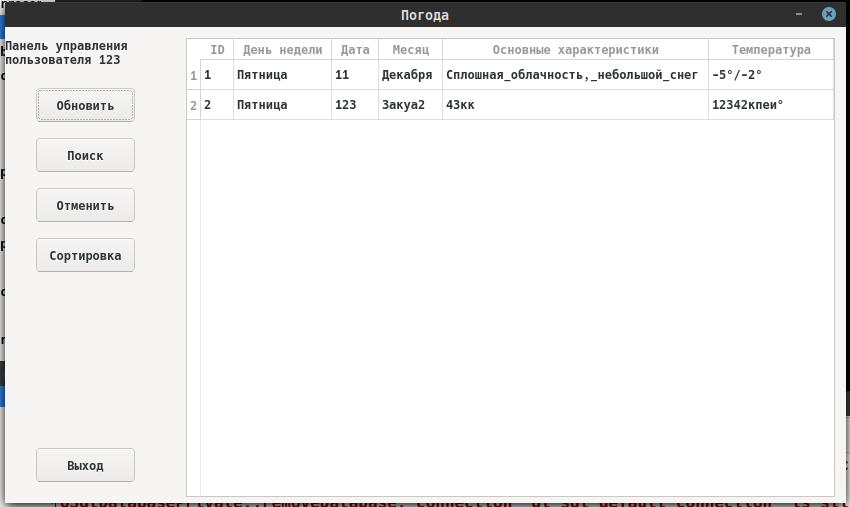


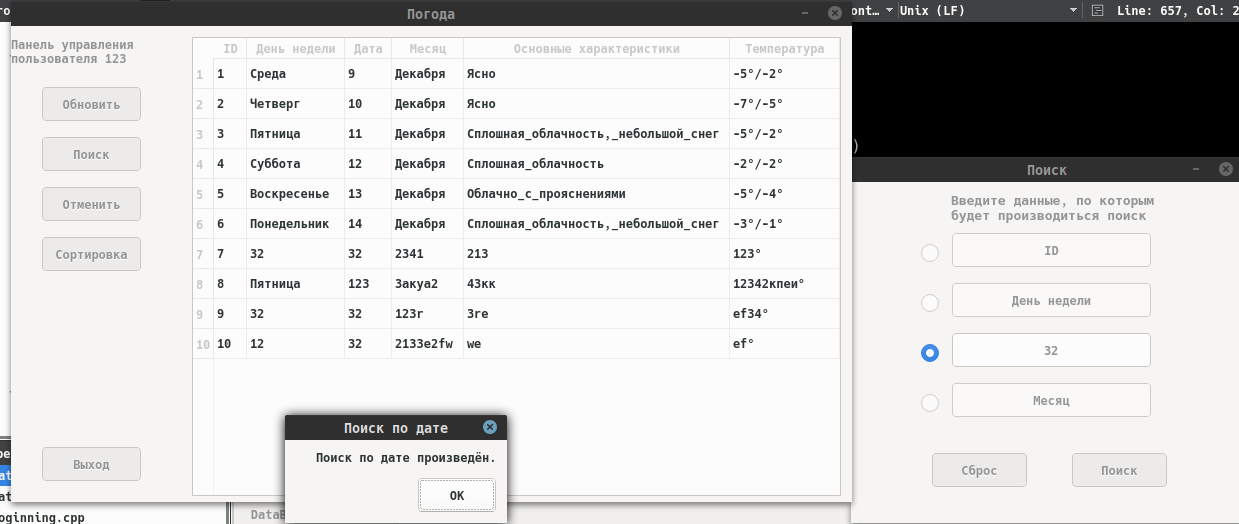
**Рис. 32 –** “Произвдена отмена предыдущих действий”

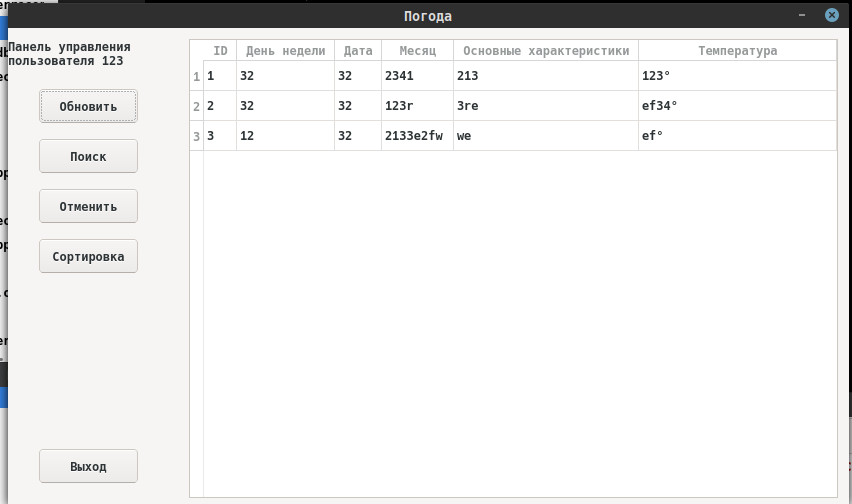
**Рис. 33 –** “Просмотр базы данных после отмены предыдущих действий”

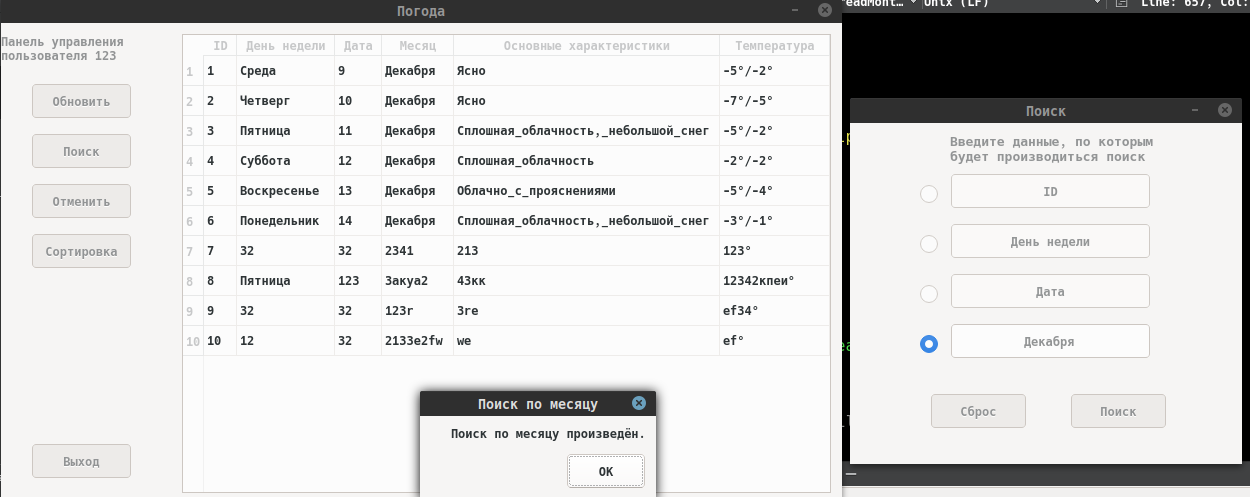
Далее я произвел некоторые изменения с базой данных с погодой, чтобы продемонстрировать работоспособность методов поиска.

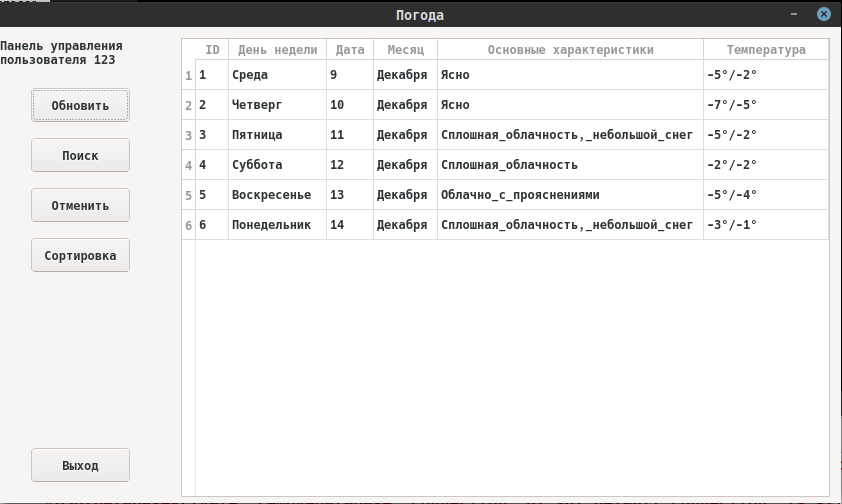
**Рис. 34 –** “Поиск по дню недели”

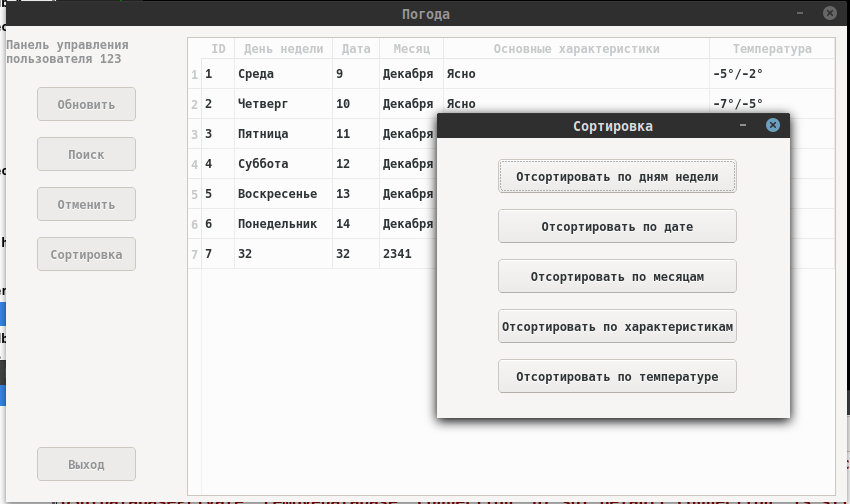
**Рис. 35 –** “Результат поиска по пятнице”

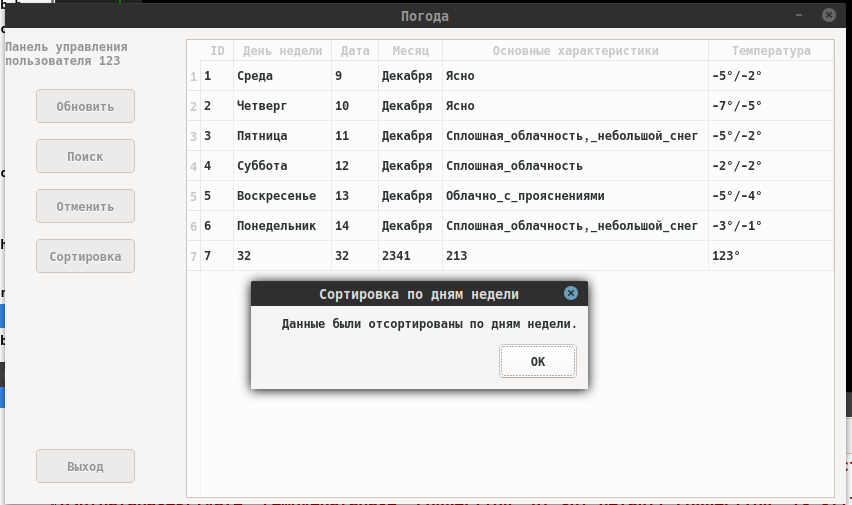
**Рис. 36 –** “Поиск по дате”

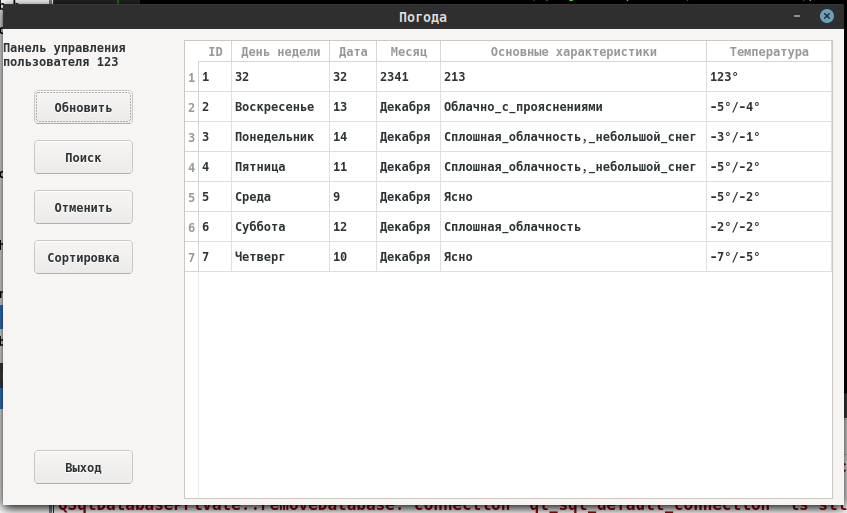
**Рис. 37 –** “Результат поиска по дате”

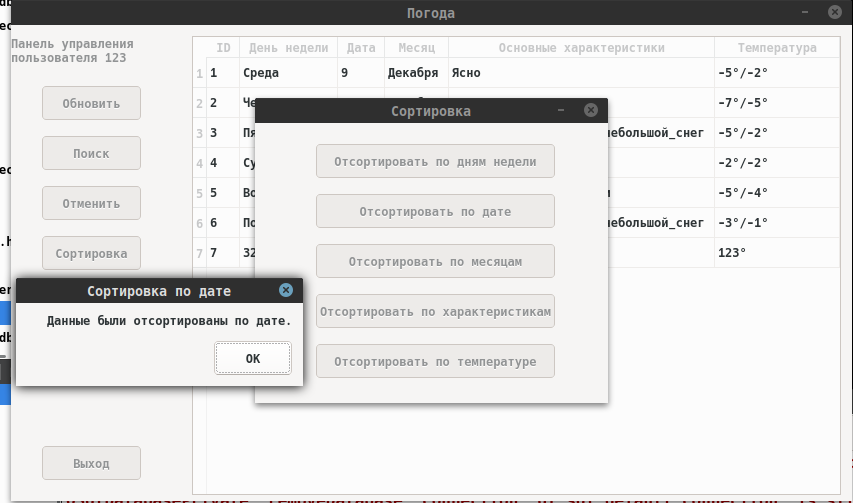
**Рис. 38 –** “Поиск по месяцу”

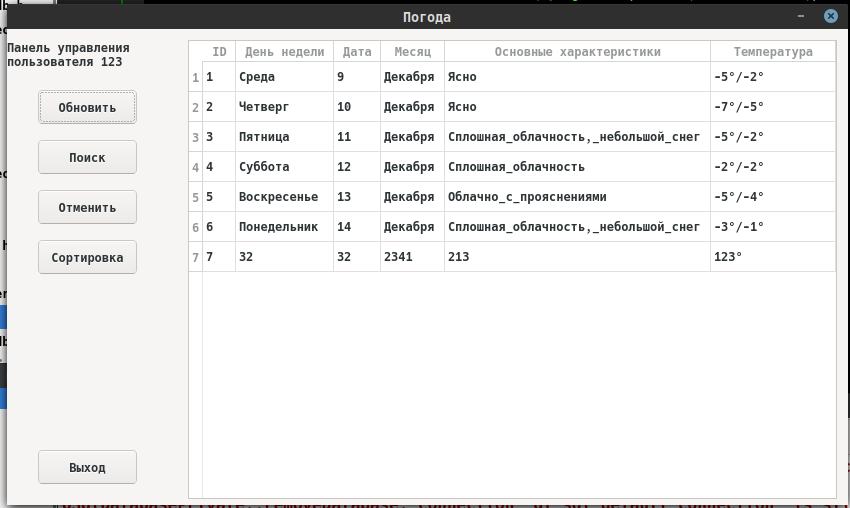
**Рис. 39 –** “Результат поиска по месяцу”

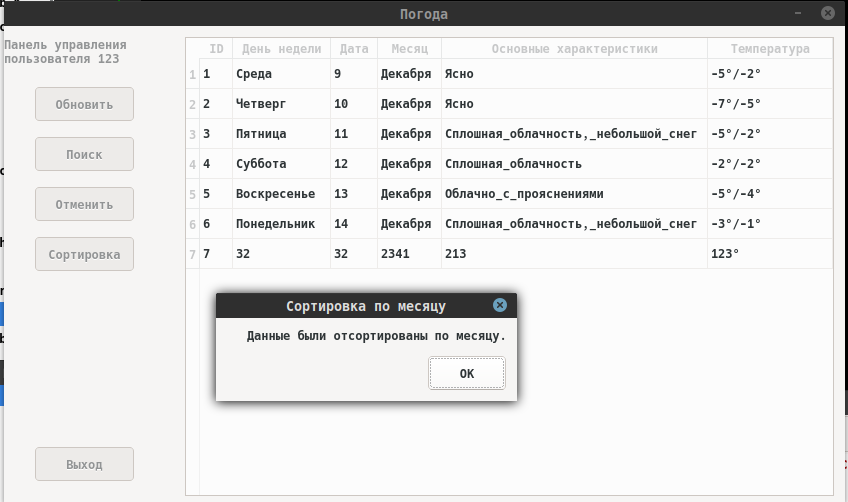
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 40 –** “Нажав на кнопку **Сортировка,** вызвал следующее окно”

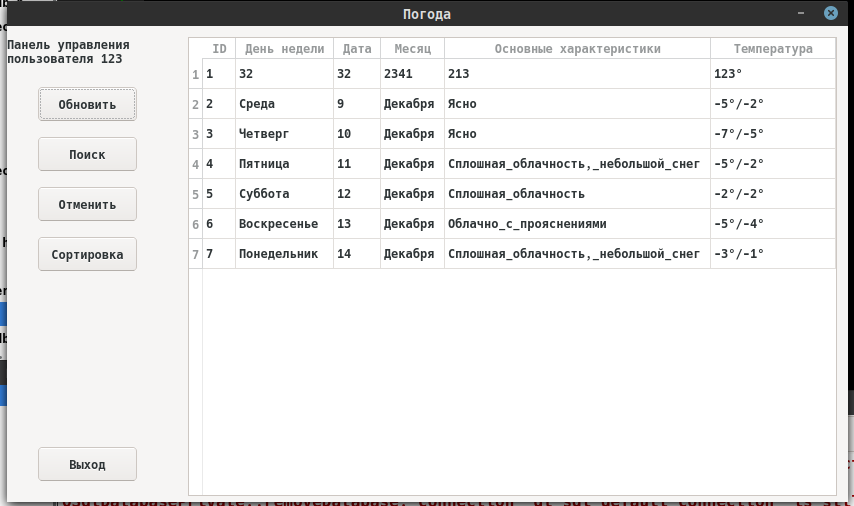
**Рис. 41 –** “Сортировка по дням недели”

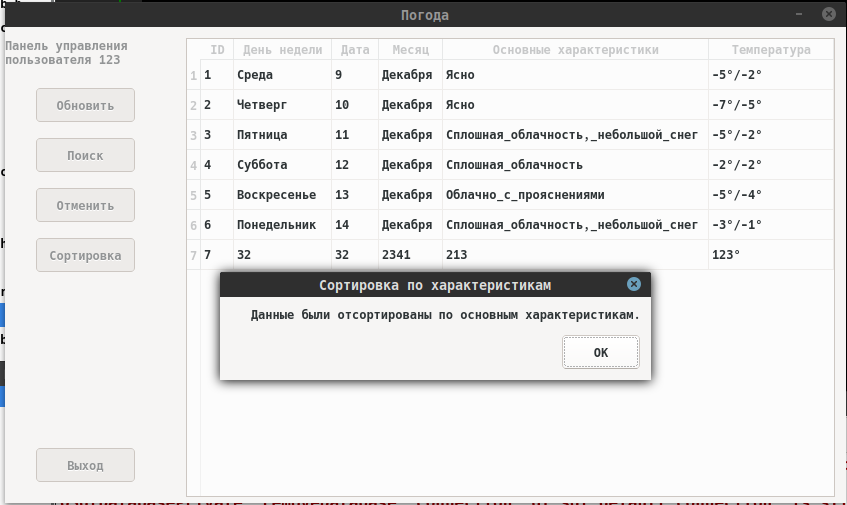
**Рис. 42 –** “Отсортированные данные по дням недели”

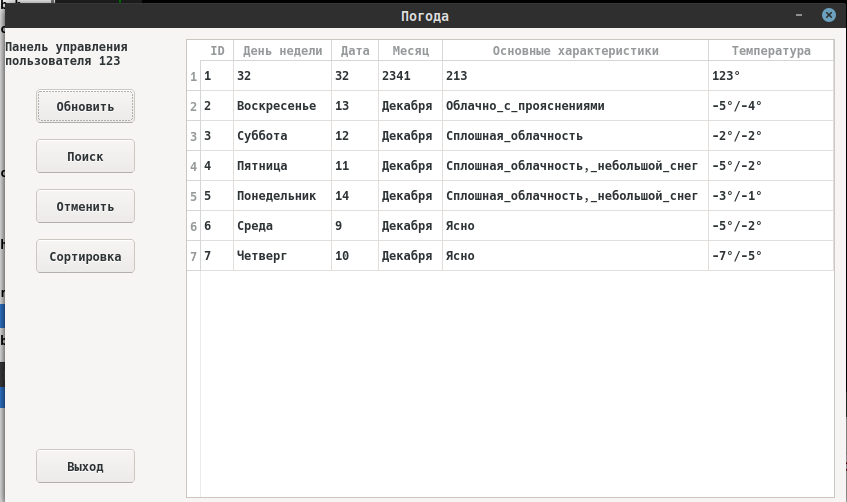
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 43 –** “Сортировка по дате”

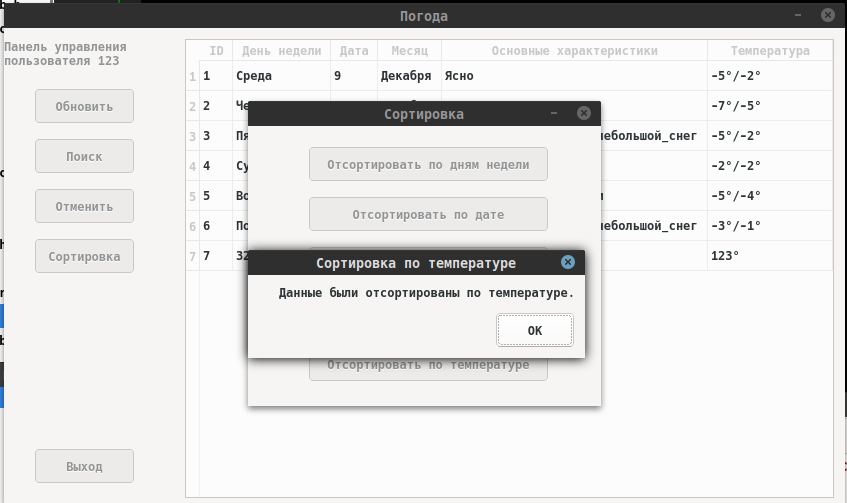
**Рис. 44 –** “Отсортированные данные по дате”

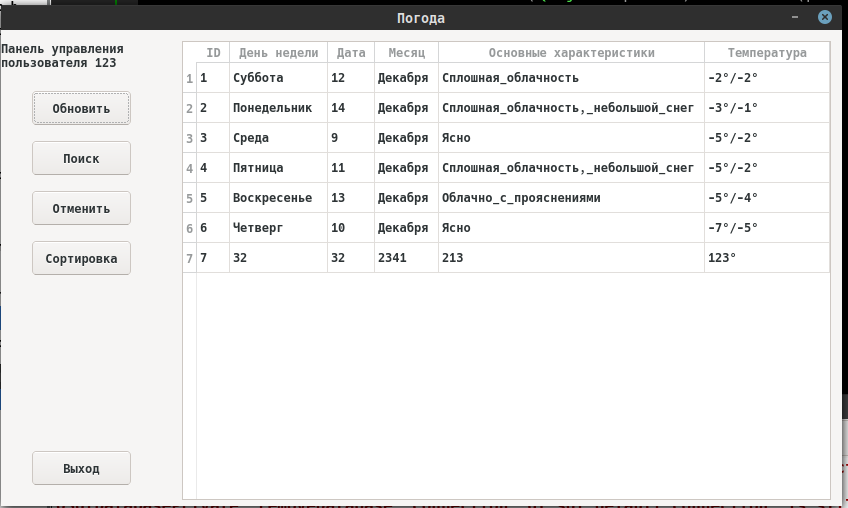
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 45 –** “Сортировка по месяцу”

**Рис. 46 –** “Отсортированные данные по месяцу”

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Рис. 47 –** “Сортировка по основным характеристикам”

**Рис. 48 –** “ Отсортированные данные по основным характеристикам”

**Рис. 49 –** “Сортировка по температуре”

**Рис. 50 –** “ Отсортированные данные по температуре”