ЛАБОРАТОРНАЯ РАБОТА №6

«ИССЛЕДОВАНИЕ СПОСОБОВ РАБОТЫ С БАЗАМИ ДАННЫХ В QT-ПРИЛОЖЕНИЯХ»

1. Цель работы

Исследование способов взаимодействия с базами данных в Qt-приложениях. Приобретение навыков разработки приложений на основе баз данных на примере SQLite.

2. Постановка задачи

2.1. Изучить принципы работы с базами данных в Qt, способы соединения с БД, способы выполнения SQL-запросов (выполняется в ходе самостоятельной подготовки к лабораторной работе).

2.2. Установить SQLite. Выполнить скрипт из приложения Б для создания таблицы и добавления тестовых записей.

2.3. Создать Qt GUI приложение.

2.4. В дизайнере добавить на форму QTableView и необходимые элементы управления.

2.5. Реализовать логику приложения по варианту задания (Приложение В).

2.6. Исследовать работу созданного приложения, проанализировать работоспособность программы при вводе ошибочных данных.

3. Ход работы

Был установлен SQLite3 по ссылке из методических указаний. Далее открыта командная строка и введены команды, изображенные на рисунке 1.

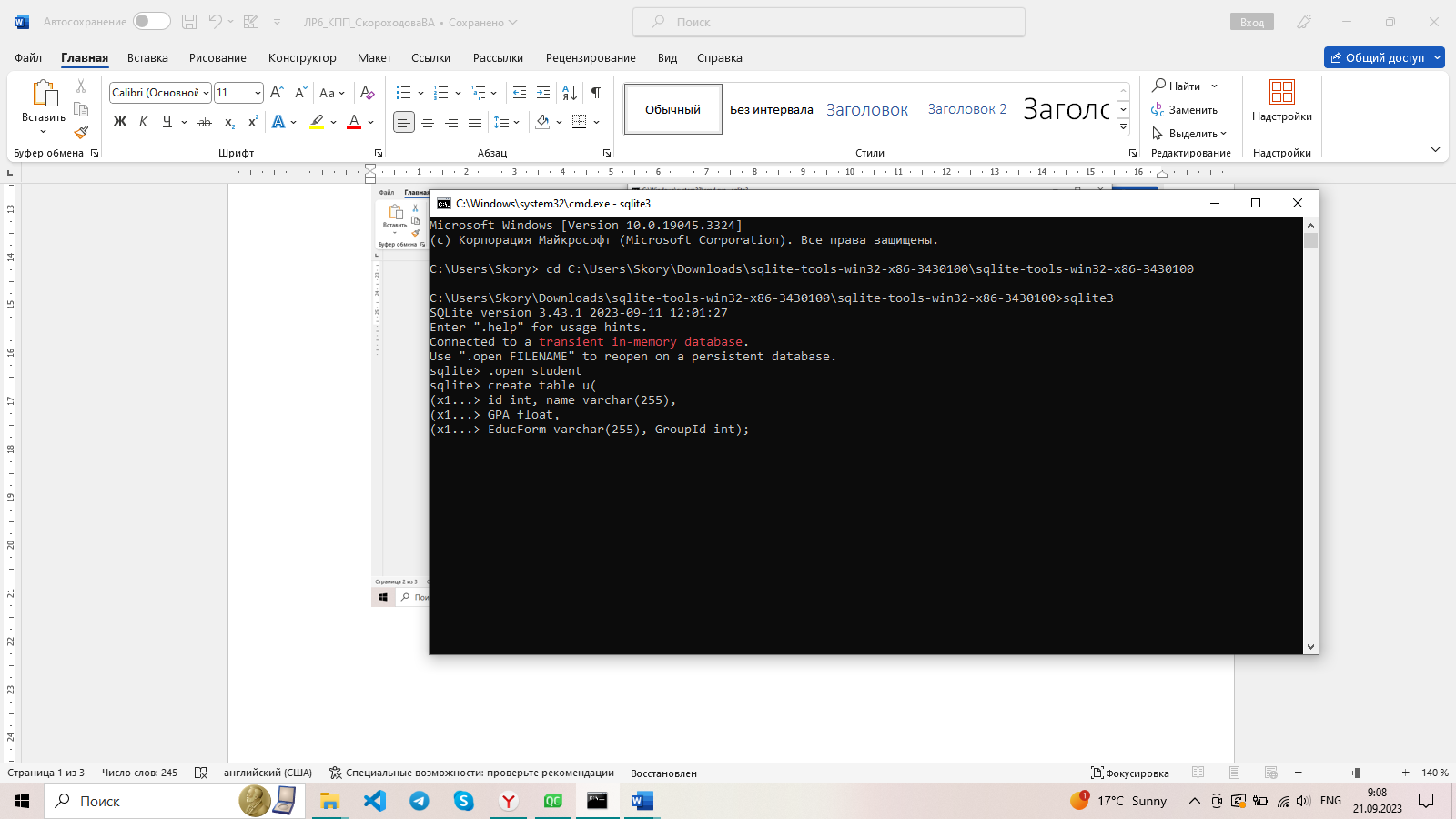


Рисунок 1 – Запуск SQLite и создание БД

Сначала с помощью команды cd произошло перемещение в папку, где находилась утилита SQLite3.

Затем с использованием команды .open student была создана БД «student.db».

Наконец в БД была добавлена таблица u с колонками id типа int, name varchar и т.д.

После создания таблицы был введен скрипт из приложения Б методических указаний. Таким образом было добавлено в таблицу u 26 записей о студентах. Скрипт представлен в листинге 1.

Листинг 1 – Скрипт на вставку 26 записей в таблицу БД

INSERT INTO U VALUES (1, 'Ivanov Ivan', 4.1, 'Zaochnaja', 'I33-z');

INSERT INTO U VALUES (2, 'Petr Pervyj', 4.3, 'Zaochnaja', 'M24-z');

INSERT INTO U VALUES (3, 'Ovchinnikova Svetlana', 4.8,'Ochnaja', 'I52-d');

INSERT INTO U VALUES (4, 'Zhelenkov Oleg', 4.4, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (5, 'Zhilin Andrej', 4.8, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (6, 'Lunjov Dmitrij', 4.2, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (7, 'Belous Tatjana', 4.5, 'Ochnaja', 'I52-d');

INSERT INTO U VALUES (8, 'Opanashhenko Irina', 4.8, 'Ochnaja','I52-d');

INSERT INTO U VALUES (9, 'Neponjatnyj Tip', 3.1, 'Zaochnaja','I33-z');

INSERT INTO U VALUES (10, 'Sjusjukajlo Dmitrij', 3.8, 'Ochnaja','I51-d');

INSERT INTO U VALUES (11, 'Margaza Artjom', 4.0, 'Ochnaja','I52-d');

INSERT INTO U VALUES (12, 'Melnikov Oleg', 4.3, 'Ochnaja', 'I52-d');

INSERT INTO U VALUES (13, 'Krikunenko Dmitrij', 3.9, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (14, 'Jakunina Anastasija', 5.0, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (15, 'Trishina Elena', 4.4, 'Ochnaja', 'I51-d');

INSERT INTO U VALUES (16, 'Mazurenko Elena', 3.7, 'Ochnaja','I52-d');

INSERT INTO U VALUES (17, 'Adzhigeldieva Nurie', 4.1, 'Ochnaja','I52-d');

INSERT INTO U VALUES (18, 'Sidorov Denis', 3.0, 'Zaochnaja','I33-z');

INSERT INTO U VALUES (19, 'Golovach Elena', 4.1, 'Zaochnaja','M24-z');

INSERT INTO U VALUES (20, 'Kuzmenko Dmitrij', 4.7, 'Zaochnaja','I33-z');

INSERT INTO U VALUES (21, 'Gricenko Ivan', 4.7, 'Zaochnaja','I33-z');

INSERT INTO U VALUES (22, 'Horolich Vladimir', 4.1, 'Zaochnaja','M24-z');

INSERT INTO U VALUES (23, 'Azarov Nikolaj', 4.1, 'Zaochnaja','M24-z');

INSERT INTO U VALUES (24, 'Napoleon Bonapart', 4.1,'Zaochnaja', 'M24-z');

INSERT INTO U VALUES (25, 'Petrov Petr', 4.1, 'Zaochnaja', 'I33-z');

INSERT INTO U VALUES (26, 'Koshkina Ljubov', 4.1, 'Zaochnaja','I33-z');

После этого была введена команда .quit, и утилита закрылась, а в папке, где хранится утилита, появился файл только что созданной базы данных. Данный файл был скопирован в папку проекта qt.

В проекте qt, в файле mainwindow.ui, были добавлены необходимые виджеты для взаимодействия с БД, а именно Table View, которое позволяет просматривать содержимое таблицы u, и кнопки «удалить» и «редактировать». Вид приложения в редакторе изображен на рисунке 2.

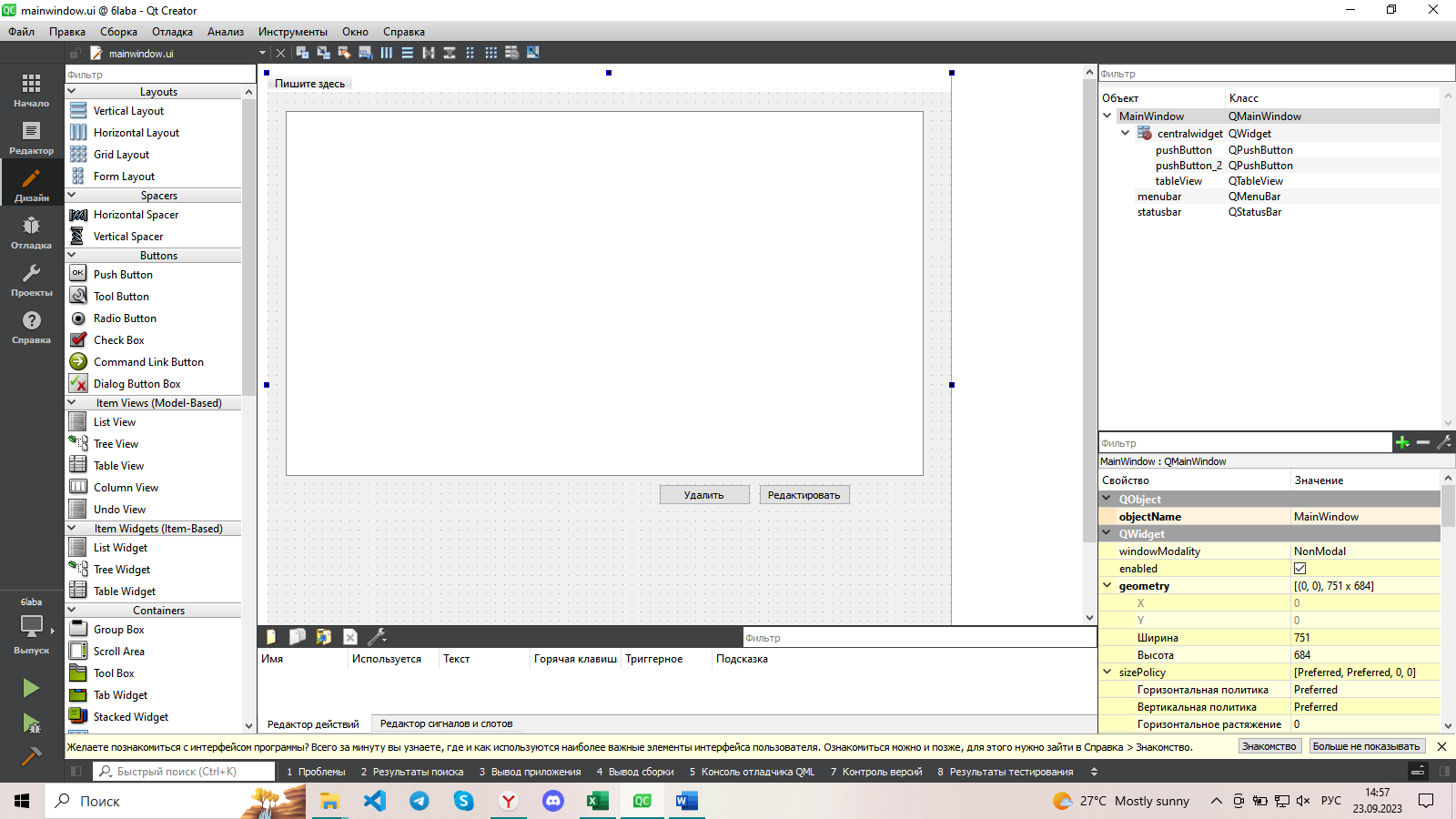


Рисунок 2 – Расположение виджетов

В листингах 1-3 представлен полный код проекта, позволяющий просматривать содержимое таблицы u, а также редактировать и удалять ее содержимое.

Листинг 1 – Файл mainwindow.h

#include <QMainWindow>

#include <QtSql>

#include <QMessageBox>

#include <QSqlDatabase>

#include <QSqlError>

#include <QSqlTableModel>

#include <QDebug>

#include <QModelIndex>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

public slots:

void **deleteSelected**();

void **edit**();

private:

Ui::MainWindow \*ui;

QSqlDatabase sdb;

QSqlTableModel \*model;

};

#endif // MAINWINDOW\_H

Листинг 2 – Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

//формируем путь к файлу БД

QString DBpath = "C:\\Users\\Skory\\Documents\\6laba\\student.db";

//добавляем нашу БД

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

sdb.setDatabaseName(DBpath);

//подключаемся

if(sdb.open()){

QMessageBox msgBox;

msgBox.setText("Подключилися");

msgBox.setStandardButtons(QMessageBox::Ok);

msgBox.*exec*();

}

else

{

QMessageBox::critical(this,tr("SQLite connection"), tr("Unable connect to DB."));

exit(1);

}

//создаем модель

model = new QSqlTableModel(this, sdb);

model->*setTable*("u");

//задаем режим редактирования при изменении поля

model->*setEditStrategy*(QSqlTableModel::OnFieldChange);

model->*select*();

//привязываем QTableView к модели

ui->tableView->*setModel*(model);

int rowCount = model->*rowCount*();

// Выводим количество загруженных записей в консоль

qDebug() << "Количество загруженных записей:" << rowCount;

//соединяем сигнал нажатия кнопки со слотом удаления записей

connect(ui->pushButton, SIGNAL(clicked()), SLOT(deleteSelected()));

connect(ui->pushButton\_2, SIGNAL(clicked()), SLOT(edit()));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**deleteSelected**()

{

QModelIndexList indexes = ui->tableView->selectionModel()->selection().indexes();

QSet<int> \*rowsToDelete = new QSet<int>();

//формируем список строк на удаление

for (int i = 0; i < indexes.count(); i++)

{

QModelIndex index = indexes.at(i);

rowsToDelete->insert(index.row());

}

//удаляем

QAbstractItemModel \*model = ui->tableView->model();

QSet<int>::iterator i;

for (i = rowsToDelete->begin(); i != rowsToDelete->end(); ++i)

{

model->removeRow(\*i);

}

}

void MainWindow::**edit**(){

QModelIndex currentIndex = ui->tableView->currentIndex();

// Проверяем, что индекс действителен

if (currentIndex.isValid())

{

// Включить редактирование ячеек для выбранной записи

ui->tableView->edit(currentIndex);

}

}

Листинг 3 – main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec(); }

Далее была проверена работа приложения. На рисунке 3 представлено само приложение. На рисунке 4 представлен процесс удаления и редактирования поля. На рисунке 5 изображена таблица после проведенных операций.

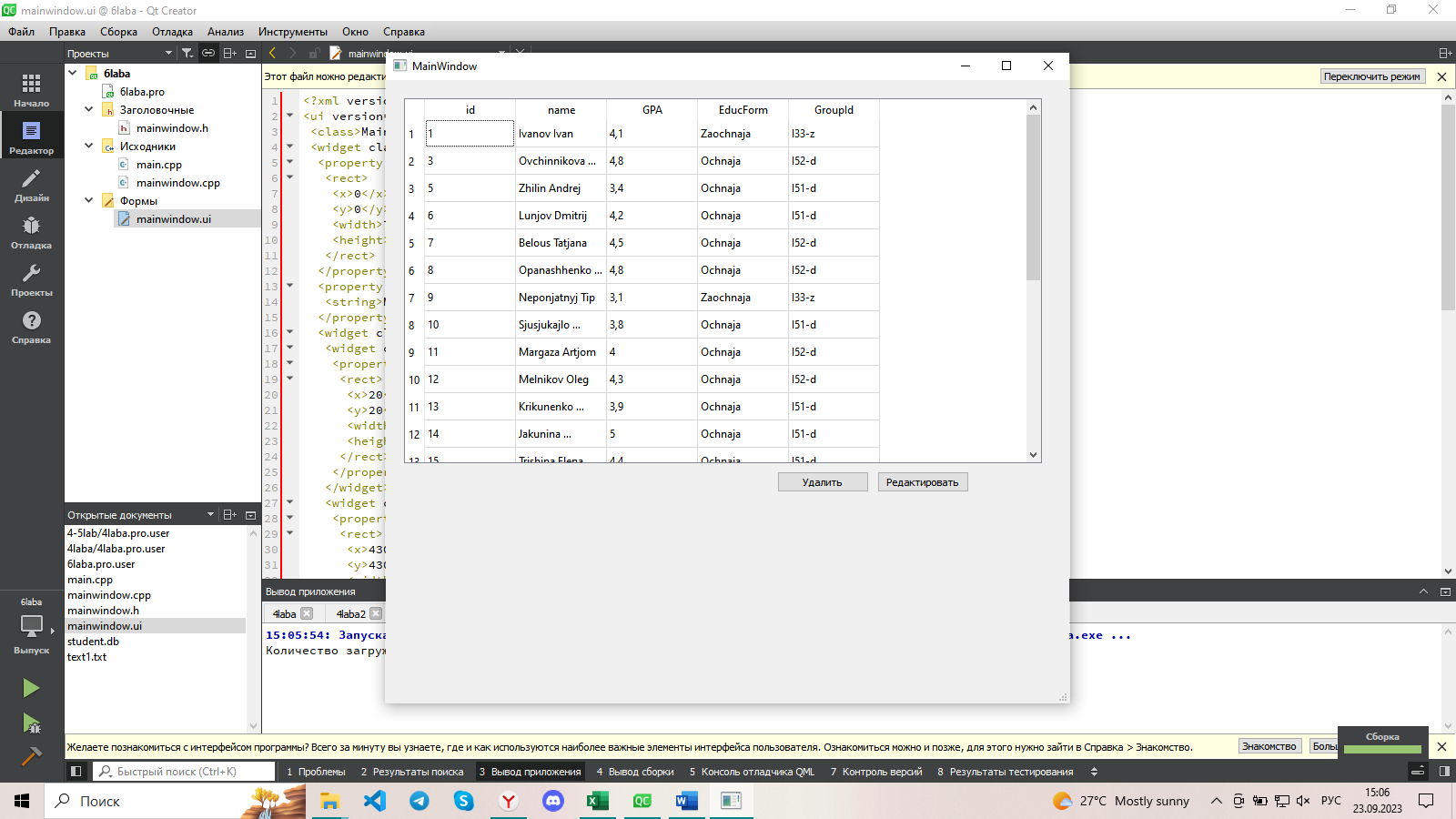


Рисунок 3 – Внешний вид приложения

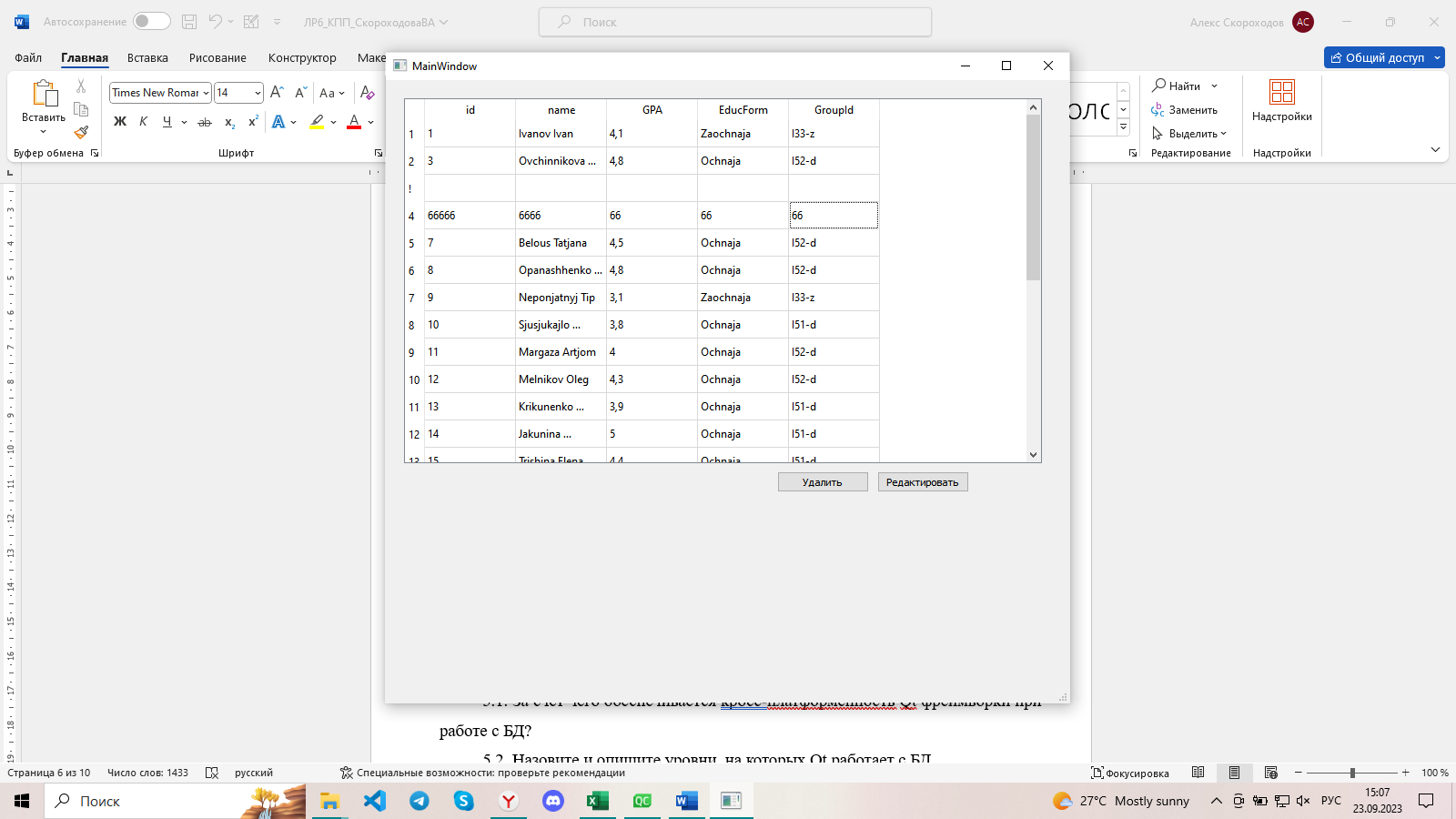


Рисунок 4 – Процесс удаления и редактирования записей

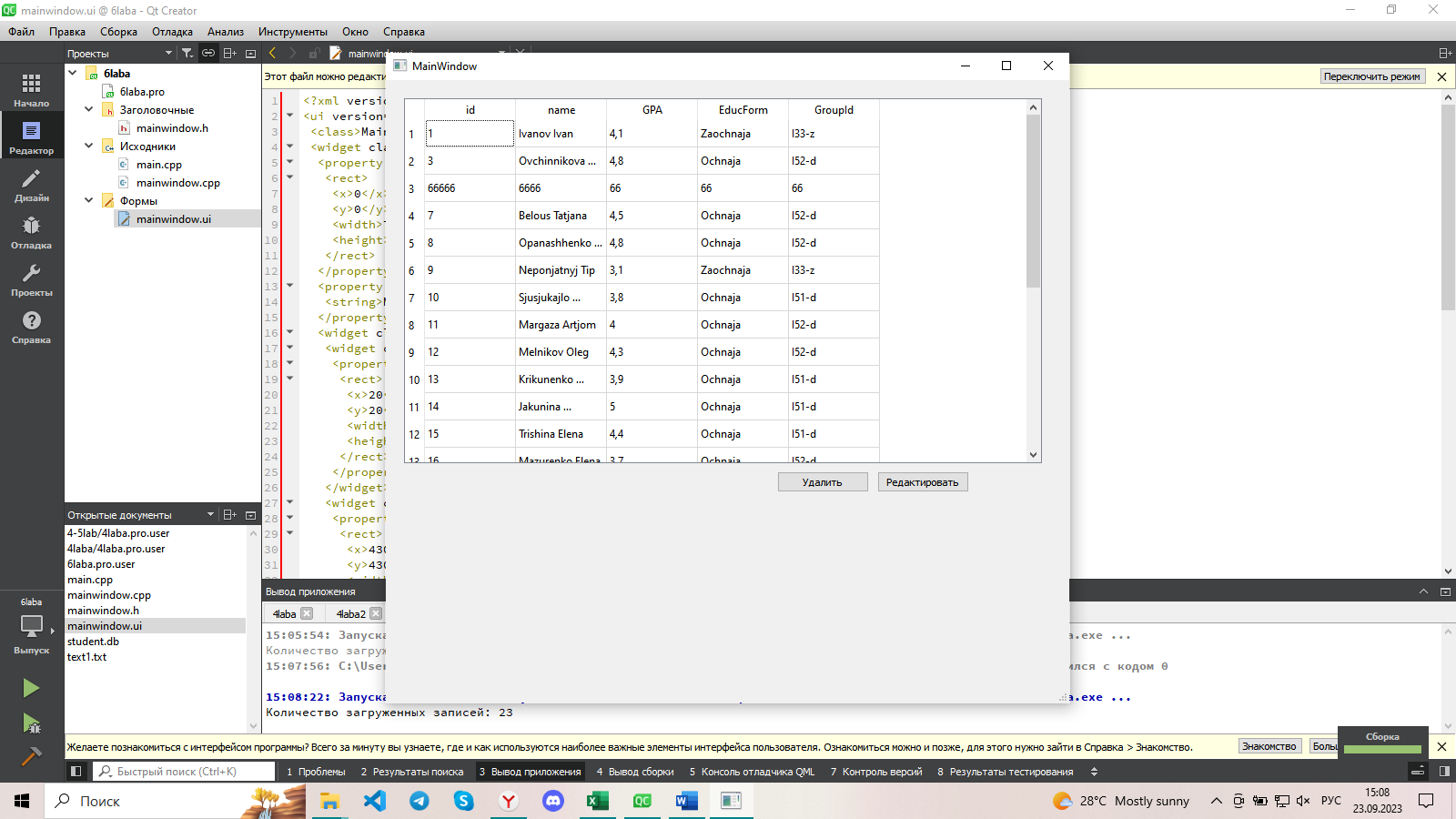


Рисунок 5 – Вид таблицы после сохраненных изменений

После тестов можно сделать выводы, что приложение работает корректно.

ВЫВОДЫ

Было проведено исследование способов взаимодействия с базами данных в Qt-приложениях, приобретены навыки разработки приложений на основе баз данных на примере SQLite.

С помощью утилиты была создана БД с таблицей, в которую были помещены несколько записей о студентах. Было написано приложение для взаимодействия и изменения готовой таблицы в БД. Программа может удалять, редактировать и сохранять изменения в БД.

Контрольные вопросы:

5.1. За счет чего обеспечивается кросс-платформенность Qt фреймворки при работе с БД?

Qt включает «родные» драйвера для Oracle, Microsoft SQL Server, Sybase Adaptive Server, IBM DB2, PostgreSQL, MySQL и ODBC-совместимых баз данных. Qt включает специфичные для баз данных виджеты, а также поддерживает расширение для работы с базами данных любых встроенных или отдельно написанных.

Кросс-платформенность Qt фреймворка при работе с базами данных достигается за счет использования абстракций, которые скрывают различия между разными базами данных и операционными системами. Qt предоставляет свой собственный интерфейс для работы с базами данных, называемый Qt SQL, который абстрагирует приложение от конкретных драйверов баз данных и позволяет писать переносимый код.

5.2. Назовите и опишите уровни, на которых Qt работает с БД.

Уровень драйвера базы данных (QSqlDriver): Он предоставляет интерфейс для взаимодействия с конкретными драйверами баз данных, такими как SQLite, MySQL, PostgreSQL и другими.

Уровень подключения к базе данных (QSqlDatabase): Он обеспечивает соединение с базой данных и управление соединениями.

Уровень выполнения SQL-запросов (QSqlQuery): Он предоставляет возможность выполнения SQL-запросов к базе данных и получения результатов.

5.3. Опишите процедуру соединения Qt к БД.

Создание экземпляра QSqlDatabase.

Установка имени драйвера базы данных (например, "QSQLITE" для SQLite).

Установка параметров соединения, таких как имя базы данных, имя пользователя, пароль и другие в зависимости от драйвера.

Вызов метода open() для установки соединения с базой данных.

Пример:

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

db.setDatabaseName("mydatabase.db");

if (db.open()) {

// Соединение установлено

} else {

// Ошибка при установке соединения

}

5.4. Каким образом возможно выполнение SQL запросов в Qt?

В Qt возможно выполнение SQL-запросов с использованием класса QSqlQuery. Для выполнения запроса необходимо создать экземпляр QSqlQuery, установить SQL-запрос с помощью метода setQuery(), а затем выполнить запрос с помощью метода exec() или получить результаты запроса.

Пример:

QSqlQuery query;

query.exec("SELECT name, salary FROM employee WHERE salary > 50000");

5.5. На каком из уровней реализуется выполнение SQL-запросов в Qt?

Выполнение SQL-запросов в Qt реализуется на уровне QSqlQuery, который использует драйвер базы данных (QSqlDriver) для выполнения запросов и получения результатов.

5.6. Опишите, каким образом можно привязать таблицу-представление к таблице БД?

Для привязки таблицы-представления к таблице БД в Qt можно использовать класс QSqlTableModel. Он предоставляет модель данных для таблицы БД и автоматически обновляет представление (например, QTableView) при изменении данных в таблице.

Пример:

QSqlTableModel model;

model.setTable("mytable");

model.select(); // Получение данных из таблицы

tableView->setModel(&model);

5.7. Поясните назначение классов QSqlDriver, QSqlDriverCreator, QSqlDriverCreatorBase, QSqlDriverPlugin и QSqlResult.

5.8. Поясните назначение классов QSqlDatabase и QsqlQuery.

5.9. Какие компоненты используются для выполнения SQL-запросов в Qt?

5.10. Работу с какими СУБД поддерживает Qt?

5.7. Классы связанные с работой с базами данных в Qt:

QSqlDriver: Предоставляет интерфейс для взаимодействия с конкретным драйвером базы данных.

QSqlDriverCreator: Абстрактный класс, используемый для создания экземпляров QSqlDriver.

QSqlDriverCreatorBase: Базовый класс для конкретных создателей драйверов баз данных.

QSqlDriverPlugin: Плагин, который предоставляет создателей драйверов баз данных.

QSqlResult: Представляет результат выполнения SQL-запроса.

5.8. Класс QSqlDatabase предоставляет соединение с базой данных и управление соединиями. Он позволяет устанавливать параметры соединения, открывать и закрывать соединения с базой данных.

Класс QSqlQuery используется для выполнения SQL-запросов к базе данных и получения результатов. Он предоставляет методы для установки и выполнения запросов, а также для извлечения данных из результата.

5.9. Для выполнения SQL-запросов в Qt используются следующие компоненты:

QSqlDatabase: Предоставляет соединение с базой данных и управление соединениями.

QSqlQuery: Используется для выполнения SQL-запросов к базе данных и получения результатов.

QSqlTableModel: Предоставляет модель данных для таблицы базы данных и автоматически обновляет представления при изменении данных.

QSqlQueryModel: Предоставляет модель данных для выполнения SQL-запросов и отображения результатов.

Драйверы баз данных (например, QSQLITE, QMYSQL, QPSQL): Реализуют взаимодействие с конкретными базами данных.

5.10. Qt поддерживает работу с различными системами управления базами данных (СУБД), включая:

SQLite

MySQL

PostgreSQL

Oracle

Microsoft SQL Server