**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Отчет

По лабораторной работе №6

На тему: “Исследование способов работы с базами данных в QT-приложениях”

По дисциплине: “Кроссплатформенное программирование”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Строганов В.А.

Севастополь

2021

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.

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.

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

Вариант 1

Отобразить таблицу Student без возможности редактирования и удаления записей. Реализовать функциональность добавления новых студентов.

3 ТЕКСТ ПРОГРАММЫ

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtSql>

#include <QMessageBox>

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 **addStudent**();

private:

Ui::MainWindow \*ui;

QSqlDatabase sdb;

};

#endif // MAINWINDOW\_H

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QString DBpath = QDir::toNativeSeparators(qApp->applicationDirPath() + "/student.db");

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

sdb.setDatabaseName(DBpath);

if(!sdb.open())

{

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

exit(1);

}

QSqlTableModel \*model = new QSqlTableModel(*ui->studentView*);

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

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

model->*select*();

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

connect(ui->buttonAdd, SIGNAL(clicked()), SLOT(addStudent()));

}

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

{

delete ui;

}

void MainWindow::**addStudent**()

{

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

if (ui->lineEdit->text() == "" ||

ui->lineEdit\_2->text() == "" ||

ui->lineEdit\_3->text() == "" ||

ui->lineEdit\_4->text() == "")

{

QMessageBox msgBox;

msgBox.setWindowTitle("Ошибка");

msgBox.setText("Заполните все поля!");

msgBox.setStandardButtons(QMessageBox::Ok);

msgBox.*exec*();

}

else

{

model->insertRow(model->*rowCount*());

int rowCount = model->*rowCount*() - 1;

model->*setData*(model->*index*(rowCount, 0), model->*rowCount*());

model->*setData*(model->*index*(rowCount, 1), ui->lineEdit->text());

model->*setData*(model->*index*(rowCount, 2), ui->lineEdit\_2->text());

model->*setData*(model->*index*(rowCount, 3), ui->lineEdit\_3->text());

model->*setData*(model->*index*(rowCount, 4), ui->lineEdit\_4->text());

model->*submit*();

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_3->setText("");

ui->lineEdit\_4->setText("");

}

}

4 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Войдем в приложение, интерфейс которого представлен на рисунке 1, и нажмем “Добавить” не вводя данных, получим окно с ошибкой (рисунок 2).

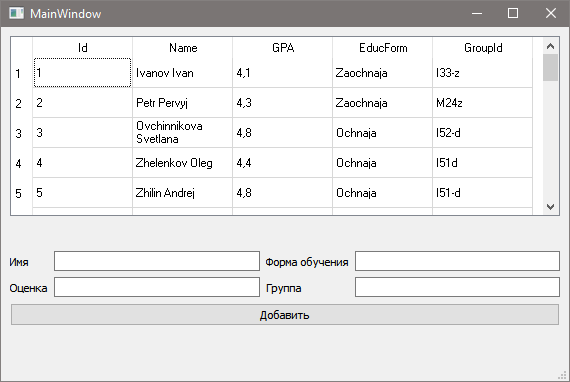


Рисунок 1 – Интерфейс приложения

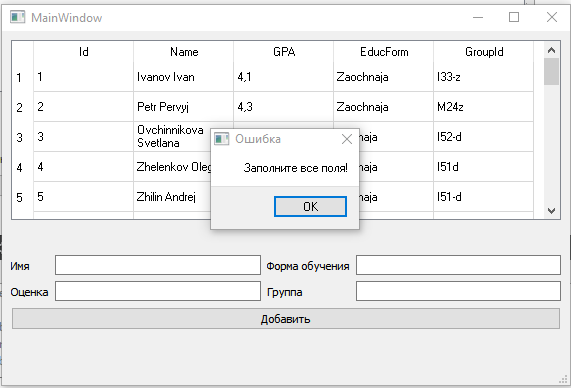


Рисунок 2 – Диалоговое окно с ошибкой

Теперь введем корректные данные и добавим в таблицу, результат добавления представлен на рисунке 3.

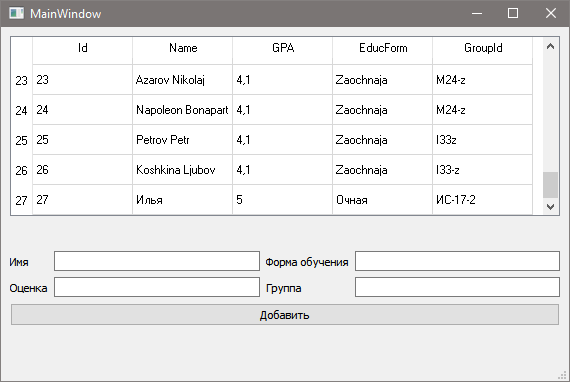


Рисунок 3 – Результат добавления данных

ВЫВОДЫ

В ходе выполнения лабораторной работы было исследовано взаимодействие с базами данных в Qt-приложениях.

Qt дает возможность создания платформо-независимых приложений для работы с базами данных, используя стандартные СУБД. 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.

Работа с базами данных в Qt происходит на различных уровнях:

1.Слой драйверов — включает классы QSqlDriver, QSqlDriverCreator, QSqlDriverCreatorBase, QSqlDriverPlugin и QSqlResult.

2.Слой SQL API — этот слой предоставляет доступ к базам данных. Соединения устанавливаются с помощью класса QSqlDatabase. Взаимодействие с базой данных осуществляется с помощью класса QSqlQuery.

3.Слой пользовательского интерфейса — этот слой связывает данные из базы данных с дата-ориентироваными виджетами (QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel).