Специальность: 09.02.07 Информационные системы и программирование

Проверил:

\_\_\_\_\_\_\_\_\_\_\_ Антонов М.Б.

Дата: «28» марта 2020 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

(Университет ИТМО)

факультет среднего профессионального образования

Отчет

о Лабораторной работе №4

по теме: Работа с базой данных

по дисциплине: Разработка программных модулей

Выполнил:

студент группы 2333

\_\_\_\_\_\_\_\_\_\_\_ Агапова Е.А.

Санкт-Петербург 2020

Содержание

[1 Постановка задачи 3](#_Toc35698558)

[1.1 Цель 3](#_Toc35698559)

[1.2 Задачи 3](#_Toc35698560)

[1.3 Задание 3](#_Toc35698561)

[2 Диаграммы 5](#_Toc35698562)

[3 Исходный код 7](#_Toc35698563)

[4 ЛИСТИНГ ВЫполнения программы 13](#_Toc35698564)

[5 Вывод 15](#_Toc35698565)

1. Постановка задачи
   1. Цель

Познакомиться с основами работы с базами данных в приложениях Qt.

* 1. Задачи

1. изучить основы работы с БД;
2. создать простейшее приложение для сохранения/обновления данных в БД.
   1. Задание

Программа должна уметь:

* соединяться с базой данных (SQLite);
* если БД не существует, то приложение должно её создать;
* если таблиц, с которыми можно работать не найдено, то приложение должно автоматически создать эти таблицы.

Требуется реализовать функцию построчного редактирования двух таблиц. Примерный интерфейс представлен на рисунке 1.

* add позволяет создать новую запись;
* save сохраняет изменения;
* delete удаляет запись;



Рисунок 1 – Пример интерфейса

Требуется реализовать просмотр 2-3 результатов запроса через QTableView.

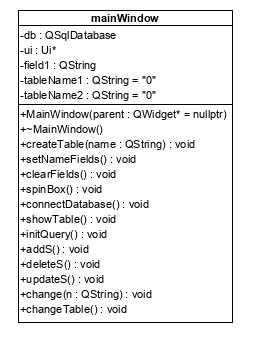
Общее ограничение на лабораторную работу:

* использовать разбитие на файлы основной программы (в main.cpp только функция main.cpp);
* документировать весь код используя аннотации Doxygen.

Требования к отчёту:

* цели лабораторной работы;
* задачи лабораторной работы;
* выполняемое задание;
* диаграмма(ы) классов;
* диаграммы деятельности (для методов и/или функций больше 10 строк кода (исключая форматирования и пустые строки));
* исходный код программы с комментариями (2 пробела вместо 4, 10pt, моноширинный шрифт);
* листинг выполнения программы;
* исходный код в виде файлов для запуска на произвольном ПК.

1. Диаграммы



*Рисунок 2 – Диаграмма классов*



*Рисунок 3 – Диаграмма деятельности MainWindow из mainwindow.cpp*

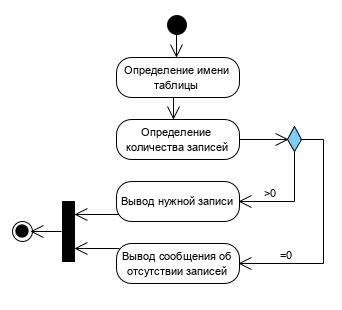


Рисунок 4 – Диаграмма деятельности change из mainwindow.cpp

1. Исходный код

Листинг 2.1 – Код программы

/\*\*

\* @author Agapova Ekaterina

\* @file main.cpp

\*/

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Листинг 2.2 – Заголовочный файл mainwindow.h

/\*\*

\* @file mainwindow.h

\* @author Agapova Ekaterina

\* @brief Work with database and ui

\*/

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QSqlDatabase>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

private:

QSqlDatabase db;

Ui::MainWindow \*ui;

QString field;

QString tName1 = "0", tName2 = "0";

public:

/\*\*

\* @brief Constructor

\* @param parent

\*/

MainWindow(QWidget \*parent = nullptr);

/\*\*

\*@brief Destructor

\*/

~MainWindow();

/\*\*

\* @brief Create Table

\* @param name of Table

\*/

void createTable(QString name);

/\*\*

\* @brief set name fields for table

\*/

void setNameFields();

/\*\*

\* @brief clear fields

\*/

void clearFields();

/\*\*

\* @brief Settings of spinBox

\*/

void spinBox();

public slots:

/\*\*

\* @brief connect Database

\*/

void connectDatabase();

/\*\*

\* @brief show Table

\*/

void showTable();

/\*\*

\* @brief query to database

\*/

void initQuery();

/\*\*

\* @brief add stroke to table

\*/

void addStroke();

/\*\*

\* @brief delete stroke from table

\*/

void deleteStroke();

/\*\*

\* @brief update stroke in table

\*/

void updateStroke();

/\*\*

\* @brief line-by-line editing

\* @param n - number of stroke

\*/

void change(QString n);

/\*\*

\* @brief change table

\*/

void changeTable(QString name);};

#endif // MAINWINDOW\_H

Листинг 2.3 – Заголовочный файл mainwindow.cpp

/\*\*

\* @file mainwindow.cpp

\* @author Agapova Ekaterina

\* @brief Work with database and ui

\*/

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QSqlQueryModel>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow){

ui->setupUi(this);

ui->WTable->hide();

ui->Redactor->hide();

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

connect(ui->openDB, &QPushButton::clicked, this, &MainWindow::connectDatabase);

connect(ui->showTable, &QPushButton::clicked, this, &MainWindow::showTable);

connect(ui->queryTable, &QPushButton::clicked, this, &MainWindow::initQuery);

connect(ui->add, &QPushButton::clicked, this, &MainWindow::addStroke);

connect(ui->delete\_2, &QPushButton::clicked, this, &MainWindow::deleteStroke);

connect(ui->save, &QPushButton::clicked, this, &MainWindow::updateStroke);

connect(ui->spinBox, &QSpinBox::textChanged, this, &MainWindow::change);

connect(ui->Table, &QComboBox::currentTextChanged, this, &MainWindow::changeTable);

}

MainWindow::~MainWindow() {

db.close();

delete ui;

}

void MainWindow::connectDatabase(){

db.setDatabaseName(ui->lineEdit->displayText());

if (db.open()){

ui->WTable->show();

ui->Database->hide();

}

else {

qDebug() << "Невозможно открыть БД";

return;

}

}

void MainWindow::showTable(){

QSqlQuery q;

QSqlTableModel \*table = new QSqlTableModel(parent(),db);

if (!q.exec("SELECT \* FROM " + ui->Table->currentText())) {

createTable(ui->Table->currentText());

} else {

table->setTable(ui->Table->currentText());

table->select();

ui->tableView->setModel(table);

}

}

void MainWindow::initQuery(){

QSqlQuery q;

QString select = "SELECT " + ui->entFields->displayText() + " FROM " + ui->comboBox->currentText();

if (ui->where->displayText() != "")

select+=" WHERE " + ui->where->displayText();

if (q.exec(select)) {

QSqlQueryModel \*query = new QSqlQueryModel;

query->setQuery(select);

ui->tableView->setModel(query);

} else {

qDebug() << "Некорректный запрос";

}

}

void MainWindow::createTable(QString name){

ui->Redactor->show();

QSqlQuery q;

if(tName1 == "0"){

tName1 = name;

q.exec("CREATE TABLE " + name + " (sID TEXT PRIMARY KEY NOT NULL, sName TEXT NOT NULL, GPA NUMERIC NOT NULL)");

setNameFields();

} else if(tName2 == "0"){

tName2 = name;

q.exec("CREATE TABLE " + name + " (cName TEXT PRIMARY KEY NOT NULL, state TEXT NOT NULL, enrollment INTEGER NOT NULL)");

setNameFields();

clearFields();

} else

q.exec("CREATE TABLE " + name + " (id INT PRIMARY KEY AUTOINCREMENT NOT NULL)");

}

void MainWindow::addStroke(){

QString name = ui->Table->currentText();

QSqlQuery query;

if(name == tName1){

query.prepare("INSERT INTO " + name + "(sID, sName, GPA) VALUES (:sID, :sName, :GPA);");

query.bindValue(":sID", ui->field1->displayText());

query.bindValue(":sName", ui->field2->displayText());

query.bindValue(":GPA", ui->field3->displayText());

} else if (name == tName2) {

query.prepare("INSERT INTO " + name + "(cName, state, enrollment) VALUES (:cName, :state, :enrollment);");

query.bindValue(":cName", ui->field1->displayText());

query.bindValue(":state", ui->field2->displayText());

query.bindValue(":enrollment", ui->field3->displayText());

}

if(query.exec()){

qDebug() << "Строка добавлена";

spinBox();

}

}

void MainWindow::deleteStroke(){

QString name = ui->Table->currentText();

QSqlQuery query;

if(name == tName1){

query.prepare("DELETE FROM " + name + " WHERE sID = :sID AND sName = :sName AND GPA = :GPA");

query.bindValue(":sID", ui->field1->displayText());

query.bindValue(":sName", ui->field2->displayText());

query.bindValue(":GPA", ui->field3->displayText());

} else if (name == tName2) {

query.prepare("DELETE FROM " + name + " WHERE cName = :cName AND state = :state AND enrollment = :enrollment");

query.bindValue(":cName", ui->field1->displayText());

query.bindValue(":state", ui->field2->displayText());

query.bindValue(":enrollment", ui->field3->displayText());

}

if(query.exec()){

qDebug() << "Строка удалена";

spinBox();

}

}

void MainWindow::updateStroke(){

QString name = ui->Table->currentText();

QSqlQuery query;

if(name == tName1){

query.prepare("UPDATE " + name + " SET sID = :sID , sName = :sName , GPA = :GPA WHERE sID = :ressID");

query.bindValue(":sID", ui->field1->displayText());

query.bindValue(":sName", ui->field2->displayText());

query.bindValue(":GPA", ui->field3->displayText());

query.bindValue(":ressID", this->field);

} else if (name == tName2) {

query.prepare("UPDATE " + name + " SET cName = :cName , state = :state , enrollment = :enrollment WHERE cName = :rescName");

query.bindValue(":cName", ui->field1->displayText());

query.bindValue(":state", ui->field2->displayText());

query.bindValue(":enrollment", ui->field3->displayText());

query.bindValue(":rescName", this->field);

}

if(query.exec())

qDebug() << "Строка отредактирована";

}

void MainWindow::change(QString n){

int i = n.toInt();

QString name = ui->Table->currentText();

QSqlQuery query;

query.exec("SELECT COUNT(\*) FROM " + name);

query.next();

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

query.exec("SELECT \* FROM " + name);

int k=0;

while(k!=i){

query.next();

k++;

}

ui->field1->setText(query.value(0).toString());

ui->field2->setText(query.value(1).toString());

ui->field3->setText(query.value(2).toString());

this->field = ui->field1->displayText();

} else {

ui->spinBox->setMinimum(0);

qDebug() << "Нет записей";

return;

}

}

void MainWindow::setNameFields(){

if(ui->Table->currentText() == tName1){

ui->nameField1->setText("sID");

ui->nameField2->setText("sName");

ui->nameField3->setText("GPA");

} else if(ui->Table->currentText() == tName2){

ui->nameField1->setText("cName");

ui->nameField2->setText("state");

ui->nameField3->setText("enrollment");

}

}

void MainWindow::clearFields(){

ui->field1->clear();

ui->field2->clear();

ui->field3->clear();

}

void MainWindow::spinBox(){

QSqlQuery q;

q.exec("SELECT COUNT(\*) FROM " + ui->Table->currentText());

q.next();

if(q.value(0).toInt() == 0)

return;

ui->spinBox->setMinimum(1);

ui->spinBox->setMaximum(q.value(0).toInt());

}

void MainWindow::changeTable(QString name){

if(name == tName1 || name == tName2){

spinBox();

setNameFields();

change("1");

}

}

1. ЛИСТИНГ ВЫполнения программы

На рисунках 5-6 представлены интерфейсы работы программы.

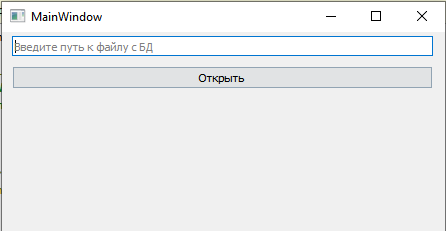


Рисунок 5 – Подключение к таблице

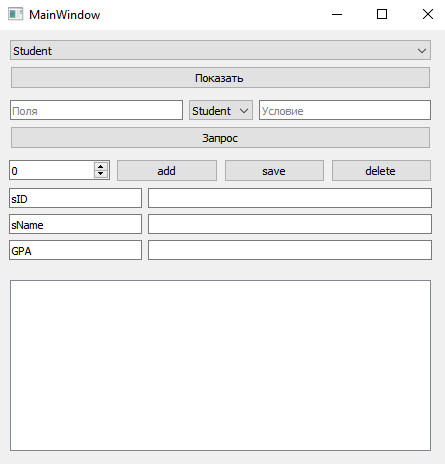


Рисунок 6 – Вывод таблиц, конструктор запросов и построчное редактирование

1. Вывод

В ходе выполнения лабораторной работы была написана программа, которая соединяется с базой данных, позволяет просматривать таблицы и отправлять запросы через конструктор запросов. Если базы данных не существует, то программа создает пустую базу данных. Если таблицы не существует, создается пустая таблица. Для некоторых таблиц реализован интерфейс построчного редактирования.