Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Севастопольский государственный университет»

ИССЛЕДОВАНИЕ СПОСОБОВ РАБОТЫ С БАЗАМИ ДАННЫХ В QT-ПРИЛОЖЕНИЯХ

Методические указания

к лабораторной работе по дисциплине

«Кроссплатформенное программирование»

для студентов, обучающихся по направлению

09.03.02 “Информационные системы и технологии”

очной и заочной форм обучения

Севастополь
2018

УДК 004.415.2
Исследование способов работы с базами данных в Qt-приложениях. Методические указания/Сост. Строганов В.А. – Севастополь: Изд-во СевГУ, 2018.–14 с.

Методические указания предназначены для оказания помощи студентам при выполнении лабораторных работ по дисциплине «Кроссплатформенное программирование».

Методические указания составлены в соответствии с требованиями программы дисциплины «Кроссплатформенное программирование» для студентов направления 09.03.02 и утверждены на заседании кафедры «Информационные системы»,
протокол № от « »_____ 2018 г.

Содержание

1. ЦЕЛЬ РАБОТЫ

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

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

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. Этот слой предоставляет низкоуровневый мост между определенными базами данных и слоем SQL API.
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. В дополнение к классам QSqlDatabase и QSqlQuery слой SQL API опирается на классы QSqlError, QSqlField, QSqlIndex и QsqlRecord.
3. **Слой пользовательского интерфейса** — этот слой связывает данные из базы данных с дата-ориентированными виджетами. Сюда входят такие классы, как QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel.

2.1. Соединение с базой данных

Чтобы получить доступ к базе данных с помощью QSqlQuery и QSqlQueryModel, необходимо создать и открыть одно или более соединений с базой данных.

Qt может работать со следующими базами данных (из-за несовместимости с GPL лицензией, не все плагины поставляются с Qt Open Source Edition):

- QDB2 — IBM DB2 (версия 7.1 и выше);
- QIBASE — Borland InterBase;
- QMYSQL — MySQL;
- QOCI — Драйвер Oracle Call Interface;
- QODBC — Open Database Connectivity (ODBC) — Microsoft SQL Server и другие ODBC-совместимые базы данных;
- QPSQL — PostgreSQL (версия 7.3 и выше);
- QSQLITE2 — SQLite версии 2;

- QSQLITE — SQLite версии 3;
- QTDS — Драйвер Sybase Adaptive Server.

Для сборки плагина драйвера, который не входит в поставку Qt нужно иметь соответствующую клиентскую библиотеку для используемой СУБД.

Соединиться с базой данных можно вот так:

```
QSqlDatabase db = QsqlDatabase::addDatabase("QMYSQL", "mydb");
db.setHostName("bigblue");
db.setDatabaseName("flightdb");
db.setUserName("acarlson");
db.setPassword("1uTbSbAs");
bool ok = db.open();
```

Первая строка создает объект соединения, а последняя открывает его. В промежутке инициализируется некоторая информация о соединении, включая имя соединения, имя базы данных, имя узла, имя пользователя, пароль.

В этом примере происходит соединение с базой данных MySQL flightdb на узле bigblue. Аргумент «QMYSQL» в addDatabase() указывает тип драйвера базы данных, чтобы использовать для соединения, а «mydb» — имя соединения.

Как только соединение установлено, можно вызвать статическую функцию QSqlDatabase::database() из любого места программы с указанием имени соединения, чтобы получить указатель на это соединение. Если не передать имя соединения, она вернет соединение по умолчанию.

Если open() потерпит неудачу, он вернет false. В этом случае, можно получить информацию об ошибке, вызвав QsqlDatabase::lastError().

Для удаления соединения с базой данных, надо сначала закрыть базу данных с помощью QSqlDatabase::close(), а затем, удалить соединение с помощью статического метода QSqlDatabase::removeDatabase().

2.2. Выполнение инструкций SQL

Класс QSqlQuery обеспечивает интерфейс для выполнения SQL запросов и навигации по результирующей выборке.

Для выполнения SQL запросов, просто создают объект QSqlQuery и вызывают QSqlQuery::exec(). Например, вот так:

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

Конструктор QSqlQuery принимает необязательный аргумент QSqlDatabase, который уточняет, какое соединение с базой данных используется. Если его не указать, то используется соединение по умолчанию. Если возникает ошибка, exec() возвращает false. Доступ к ошибке можно получить с помощью QsqlQuery::lastError().

QSqlQuery предоставляет единовременный доступ к результирующей выборке одного запроса. После вызова exec(), внутренний указатель QSqlQuery указывает на позицию перед первой записью. Если вызвать метод QSqlQuery::next() один раз, то он переместит указатель к первой записи. После

этого необходимо повторять вызов next(), чтобы получать доступ к другим записям, до тех пор пока он не вернет false. Вот типичный цикл, перебирающий все записи по порядку:

```
while (query.next()) {
  QString name = query.value(0).toString();
  int salary = query.value(1).toInt();
  qDebug() << name << salary;
}
```

QSqlQuery может выполнять не только SELECT, но также и любые другие запросы. Следующий пример вставляет запись в таблицу, используя INSERT:

```
QSqlQuery query;
query.exec("INSERT INTO employee (id, name, salary) "
     "VALUES (1001, 'Thad Beaumont', 65000)");
```

Если надо одновременно вставить множество записей, то зачастую эффективней отделить запрос от реально вставляемых значений. Это можно сделать с помощью вставки значений через параметры. Qt поддерживает два синтаксиса вставки значений: поименованые параметры и позиционные параметры. В следующем примере показана вставка с помощью поименованного параметра:

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
      "VALUES (:id, :name, :salary)");
query.bindValue(":id", 1001);
query.bindValue(":name", "Thad Beaumont");
query.bindValue(":salary", 65000);
query.exec();
```

В этом примере показана вставка с помощью позиционного параметра:

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
      "VALUES (?, ?, ?)");
query.addBindValue(1001);
query.addBindValue("Thad Beaumont");
query.addBindValue(65000);
query.exec();
```

При вставке множества записей требуется вызвать QSqlQuery::prepare() только однажды. Далее можно вызвать bindValue() или addBindValue() с последующим вызовом exec() столько раз, сколько потребуется.

2.3. Отображение данных в таблице-представлении

Классы QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel могут использоваться в качестве источников данных для классов представлений Qt, таких как QListView, QTableView и QTreeView. На практике наиболее часто используется QtableView, в связи с тем, что результирующая SQL выборка, по существу, представляет собой двумерную структуру данных.

В следующем примере создается представление, основанное на модели данных SQL:

```
QSqlTableModel model;
model.setTable("employee");
QTableView *view = new QTableView;
view->setModel(&model);
view->show();
```

Если модель является моделью для чтения-записи (например, QSqlTableModel), то представление позволяет редактировать поля. Это можно отключить с помощью следующего кода:

```
view->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

Можно использовать одну и ту же модель в качестве источника данных для нескольких представлений. Если пользователь изменяет данные модели с помощью одного из представлений, другие представления немедленно отобразят изменения.

Классы-представления для обозначения колонок наверху отображают заголовки. Для изменения текста заголовка, используется функция setHeaderData() модели. Например:

```
model->setHeaderData(0, Qt::Horizontal, QObject::tr("ID"));
model->setHeaderData(1, Qt::Horizontal, QObject::tr("Name"));
model->setHeaderData(2, Qt::Horizontal, QObject::tr("City"));
model->setHeaderData(3, Qt::Horizontal, QObject::tr("Country"));
```

2.4. Пример работы Qt с SQLite БД

2.4.1. Скачиваем SQLite command-line shell из раздела Precompiled Binaries for Windows по следующей ссылке: http://www.sqlite.org/download.html

2.4.2. Распаковываем архив, открываем из командной строки, указывая в качестве первого параметра имя файла, который будет хранить БД. Пример вызова показан на рисунке 2.1.

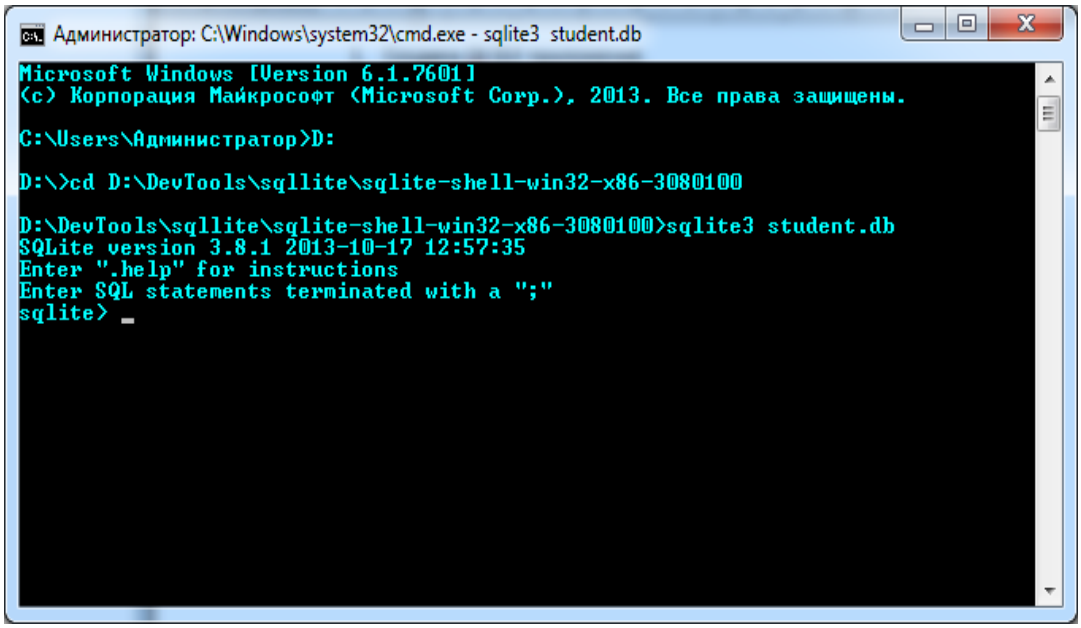

Рисунок 2.1 – Запуск SQLite

Теперь можно вводить SQL команды.

2.4.3. Для запуска SQL скрипта из файла используется команда .read <путь>

Запустим скрипт, создающий таблицу Fruit и вставляет в неё несколько записей. Текст скрипта:

```
CREATE TABLE IF NOT EXISTS Fruit
(
Id int,
Name varchar(255),
Price int
);

INSERT INTO Fruit VALUES (1, 'Banana', 100);
INSERT INTO Fruit VALUES (2, 'Orange', 150);
INSERT INTO Fruit VALUES (3, 'Pomelo', 300);
INSERT INTO Fruit VALUES (4, 'Apple', 50);
INSERT INTO Fruit VALUES (5, 'Pineapple', 500);
INSERT INTO Fruit VALUES (6, 'Cocoa', 400);
INSERT INTO Fruit VALUES (7, 'Plum', 100);
INSERT INTO Fruit VALUES (8, 'Peach', 170);
INSERT INTO Fruit VALUES (9, 'Apricot', 190);
```

Выполняем скрипт:

```
sqlite> .read D:\\a.sql
```

Обратите внимание на экранирование “\\”. При запуске скриптов из директорий, содержащих русские символы в имени, могут возникнуть проблемы.

2.4.4. Проверяем содержимое добавленной таблицы:

```
sqlite> select * from Fruit;
1¦Banana¦100
2¦Orange¦150
3¦Pomelo¦300
4¦Apple¦50
5¦Pineapple¦500
6¦Cocoa¦400
7¦Plum¦100
8¦Peach¦170
9¦Apricot¦190
sqlite>
```

Таблица и записи были добавлены корректно.

2.4.5. Завершаем работу с консольной утилитой sqlite3 командой .quit:

```
sqlite> .quit
```

В директории с консольной утилитой создался файл базы данных – student.db.

2.4.6. Создаем Qt приложение для работы с базой данных.

Приложение должно обеспечивать вывод данных из таблицы Fruit (+возможность редактирования и удаления записей). Выполняем следующие шаги:

1. Создаем Qt GUI приложение;
2. Размещаем элементы графического интерфейса пользователя. На форму помещаем виджет QTableView и кнопку, по нажатию на которую будет происходить удаление записей. Внешний вид окна приложения показан на рисунке 2.2.

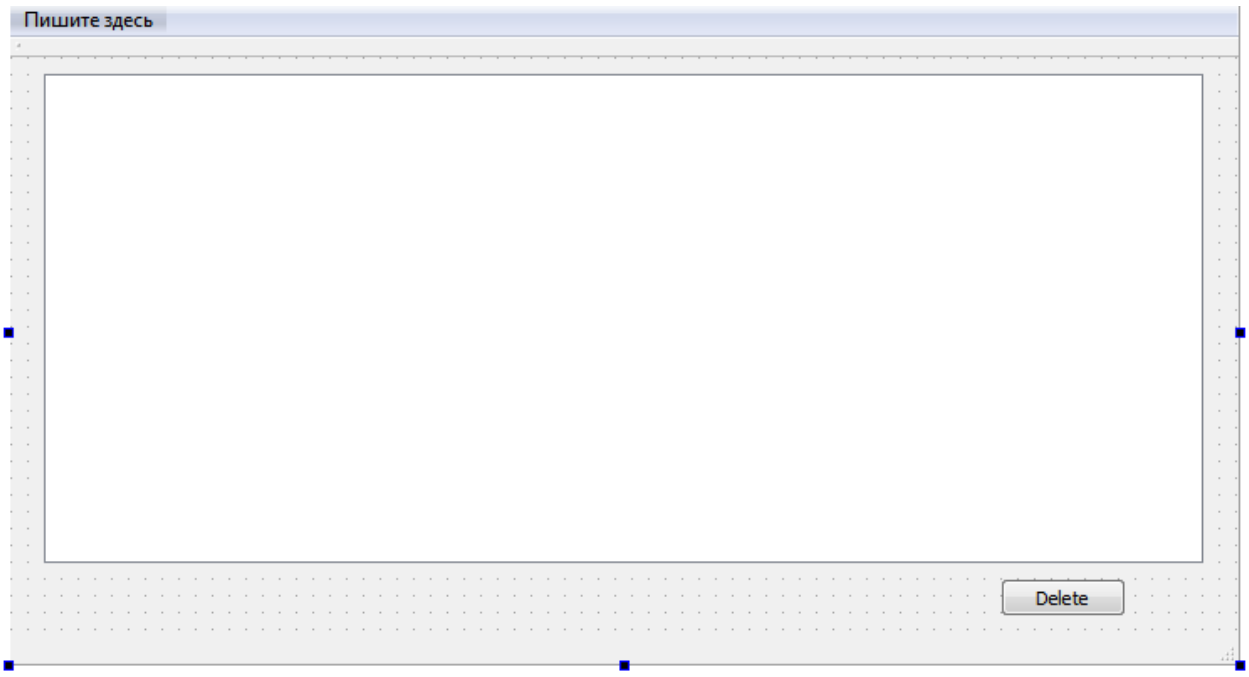

Рисунок 2.2 – Внешний вид окна приложения

3. В файл проект .pro добавляем запись: QT += sql
4. Собираем проект. В папку к созданным двоичным файлам (в директорию к lab6.exe) добавляем файл БД SQLite, созданный в пункте 2.4.2.
   Исходные тексты программных модулей приведены в приложении А.

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

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

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

3.3. Создать Qt GUI приложение.

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

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

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

4. СОДЕРЖАНИЕ ОТЧЕТА

4.1. Цель работы.

4.2. Постановка задачи.

4.3. Описание алгоритма работы приложения, внешний вид окна прилолжения.

4.4. Текст программы.

4.5. Выводы по результатам работы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

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

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

5.3. Опишите процедуру соединения Qt к БД.

5.4. Каким образом возможно выполнение SQL запросов в Qt?

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

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

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?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ж. Бланшет. Qt 3: программирование GUI на C++ / Бланшет Ж.,

Саммерфилд М. — М. : КУДИЦ — ОБРАЗ, 2005. - 448 с.

2. Е.Р. Алексеев. Программирование на языке C++ в среде Qt Creator /Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О. В. Чеснокова, А. С. Чмыхало.— М.:Альт Линукс, 2015. — 448 с.

3. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++/Г. Буч. — М. : БИНОМ ; СПб. : Невский диалект, 2001. - 560 с.

4. Шилдт, Г. C++: базовый курс, 3-е издание /Г. Шилдт. — М.: «Вильямс», 2012. — 624 с.

5. Шилдт, Г. Полный справочник по C++, 4-е издание /Г. Шилдт. — М.: «Вильямс», 2011. — 800 с.

Приложение А – Тексты программных модулей

Mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSql>
#include <QMessageBox>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
public slots:
    void deleteSelected();
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);
    //формируем путь к файлу БД
    QString DBpath = QDir::toNativeSeparators(qApp->applicationDirPath() +
"/student.db");

    //добавляем нашу БД
```

```
    sdb = QSqlDatabase::addDatabase("QSQLITE");
    sdb.setDatabaseName(DBpath);

    //пытаемся подключиться
    if(!sdb.open())
    {
    QMessageBox::critical(this, tr("SQLite connection"), tr("Unable connect to
DB, check file permission."));
    exit(1);
    }

    //создаем модель
    QSqlTableModel *model = new QSqlTableModel(ui->fruitView);
    model->setTable("Fruit");
    //задаем режим редактирования при изменении поля
    model->setEditStrategy(QSqlTableModel::OnFieldChange);
    model->select();

    //привязываем QTableView к модели
    ui->fruitView->setModel(model);

    //соединяем сигнал нажатия кнопки со слотом удаления записей
    connect(ui->delBtn, SIGNAL(clicked()), SLOT(deleteSelected()));
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::deleteSelected()
{
    //получаем индексы выделенных ячеек
    QModelIndexList indexes = ui->fruitView->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->fruitView->model();
    QSet<int>::iterator i;
    for (i = rowsToDelete->begin(); i != rowsToDelete->end(); ++i)
    {
        model->removeRow(*i);
    }
}
```

Приложение Б – Скрипт для создания таблицы БД

```
CREATE TABLE IF NOT EXISTS Student
(
Id int,
Name varchar(255),
GPA float,
```

```
EducForm varchar(255),
GroupId int
);

INSERT INTO Student VALUES (1, 'Ivanov Ivan', 4.1, 'Zaochnaja', 'I33-z');
INSERT INTO Student VALUES (2, 'Petr Pervyj', 4.3, 'Zaochnaja', 'M24-z');
INSERT INTO Student VALUES (3, 'Ovchinnikova Svetlana', 4.8, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (4, 'Zhelenkov Oleg', 4.4, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (5, 'Zhilin Andrej', 4.8, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (6, 'Lunjov Dmitrij', 4.2, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (7, 'Belous Tatjana', 4.5, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (8, 'Opanashhenko Irina', 4.8, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (9, 'Neponjatnyj Tip', 3.1, 'Zaochnaja', 'I33-z');
INSERT INTO Student VALUES (10, 'Sjusjukajlo Dmitrij', 3.8, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (11, 'Margaza Artjom', 4.0, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (12, 'Melnikov Oleg', 4.3, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (13, 'Krikunenko Dmitrij', 3.9, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (14, 'Jakunina Anastasija', 5.0, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (15, 'Trishina Elena', 4.4, 'Ochnaja', 'I51-d');
INSERT INTO Student VALUES (16, 'Mazurenko Elena', 3.7, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (17, 'Adzhigeldieva Nurie', 4.1, 'Ochnaja', 'I52-d');
INSERT INTO Student VALUES (18, 'Sidorov Denis', 3.0, 'Zaochnaja', 'I33-z');
INSERT INTO Student VALUES (19, 'Golovach Elena', 4.1, 'Zaochnaja', 'M24-z');
INSERT INTO Student VALUES (20, 'Kuzmenko Dmitrij', 4.7, 'Zaochnaja', 'I33-z');
INSERT INTO Student VALUES (21, 'Gricenko Ivan', 4.7, 'Zaochnaja', 'I33-z');
```

```
INSERT INTO Student VALUES (22, 'Horolich Vladimir', 4.1, 'Zaochnaja', 'M24-z');
INSERT INTO Student VALUES (23, 'Azarov Nikolaj', 4.1, 'Zaochnaja', 'M24-z');
INSERT INTO Student VALUES (24, 'Napoleon Bonapart', 4.1, 'Zaochnaja', 'M24-z');
INSERT INTO Student VALUES (25, 'Petrov Petr', 4.1, 'Zaochnaja', 'I33-z');
INSERT INTO Student VALUES (26, 'Koshkina Ljubov', 4.1, 'Zaochnaja', 'I33-z');
```

ПРИЛОЖЕНИЕ В – Варианты заданий

Таблица 1 – Варианты задания

Вариант	Описание
1	Отобразить таблицу Student без возможности редактирования и удаления записей. Реализовать функциональность добавления новых студентов.
2	Отобразить таблицу Student с возможностью редактирования и удаления записей напрямую из QTableView.

SQL скрипт для создания таблицы Student и добавления тестовых записей расположен в приложении Б. Следует обратить внимание на то, что утилита sqlite3 принимает скрипты только с ANSI кодировкой.

Номер варианта определяется следующим образом:

Последняя цифра зачетной книжка (вариант выданный преподавателем) % 2 + 1.

Заказ № _____ от «_____» _____ 2018 г. Тираж экз.