

Колледж космического машиностроения и технологии

**КУРСОВОЙ ПРОЕКТ**

**По дисциплине «Прикладное программирование»**

**Тема: «Учет архивных материалов»**

Выполнил студент:

Поддубный Даниил Юрьевич

Группа П3-16

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

Проверил преподаватель:

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата сдачи работы)

Королев, 2020

**Содержание**

[**Введение. 2**](#_Toc41240248)

[**Глава 1. Постановка задачи 2**](#_Toc41240249)

[**Глава 2. Проектирование. 3**](#_Toc41240250)

[**2.1 Use-case диаграмма 3**](#_Toc41240251)

[**2.2 ER диаграмма таблиц базы данных 6**](#_Toc41240252)

[**2.3 UML-диаграмма классов 7**](#_Toc41240253)

[**Глава 3. Разработка 9**](#_Toc41240254)

[**3.1 Принципы экстремального программирования 9**](#_Toc41240255)

[**3.2 Используемые сторонние библиотеки для разработки 11**](#_Toc41240256)

[**3.3 Использование системы контроля версий 12**](#_Toc41240257)

[**3.4 Описание основных классов программы 16**](#_Toc41240258)

[**Глава 4. Эксплуатация 17**](#_Toc41240259)

[**4.1 Требуемое программное и аппаратное обеспечение для работы вашего приложения. 17**](#_Toc41240260)

[**4.2 Описание установки разработанного приложения. 17**](#_Toc41240261)

[**4.3 Описание для пользователя 17**](#_Toc41240262)

[**Вывод. 23**](#_Toc41240263)

[**Библиографический список. 24**](#_Toc41240264)

[**Приложение 25**](#_Toc41240265)

# Введение.

В данной работе разрабатывается приложение для ведения учета студентов, на каких они обучаются специальностях, в каких группах состоят. Данное приложение может быть использовано в университетах на кафедрах для учета студентов.

В первой главе производится постановка задачи, выбираются подходящие технологии для ее реализации.

Во второй главе выполняется проектирование системы, разработка диаграммы реализуемых функций, разработка схемы базы данных, диаграммы классов приложения.

В третьей главе выполняется разработка приложения, описываются сторонние библиотеки, использованные в работе, описываются разработанные классы, также приводится работа с системой контроля версий.

В последней главе приводятся руководства по эксплуатации приложения, требования к использованию, руководство по установке и руководство по использованию.

В приложении к отчету имеется полный исходный код разработанной программы.

# Глава 1. Постановка задачи

Необходимо разработать приложение для ведения учета студентов. В приложении должны иметься следующие возможности: принять студента на определенную специальность, отчислить за неуспеваемость, перевести со специальности на специальность. Сформировать отчет о том, сколько студентов и на каких специальностях в данный момент обучаются. Количество студентов по группам.

В курсовой работе имеется обязательное и дополнительное задание.

Обязательное задание: разработать программное обеспечение по варианту. Для хранения информации использовать БД SQLite, для сохранения отчетов – формат pdf.

Дополнительное задание: осуществить выгрузку запрошенных данных из БД в формате XML, используя механизм XML-сериализации и вывести на форму график по заданию.

Для реализации будут использоваться Visual Studio и язык программирования C#. Приложение будет реализовано на Windows Forms, для работы с pdf и базой данных будут использоваться дополнительные библиотеки.

# Глава 2. Проектирование.

## Use-case диаграмма

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Диаграммы вариантов использования не предназначены для отображения проекта и не могут описывать внутреннее устройство системы. Диаграммы вариантов использования предназначены для упрощения взаимодействия с будущими пользователями системы, с клиентами, и особенно пригодятся для определения необходимых характеристик системы. Другими словами, диаграммы вариантов использования говорят о том, что система должна делать, не указывая сами применяемые методы.

Вариант использования описывает, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату.  
Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать [1].

Диаграмма системы приведена на Рис. 2.1. На диаграмме видны основные возможности взаимодействия пользователя с программой. Пользователь может добавлять/редактировать/удалять данные о студентах, группах и специальностях. Также имеет возможность получения представления этих данных в виде отчета pdf, дампа базы данных в формате xml и графической диаграммы.



Рис.2.1 Use-case диаграмма

## ER диаграмма таблиц базы данных

Модель сущность-связь (ER-модель) – модель данных, позволяющая описывать концептуальные схемы предметной области. ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями [2].

Сущность изображается в виде прямоугольника, содержащего её имя, выражаемое существительным. Имя сущности должно быть уникальным в рамках одной модели. При этом, имя сущности – это имя типа, а не конкретного экземпляра данного типа. Экземпляром сущности называется конкретный представитель данной сущности.

Связь изображается линией, которая связывает две сущности, участвующие в отношении. Степень конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи.

Er-диаграмма базы данных представлена на Рис. 2.2.

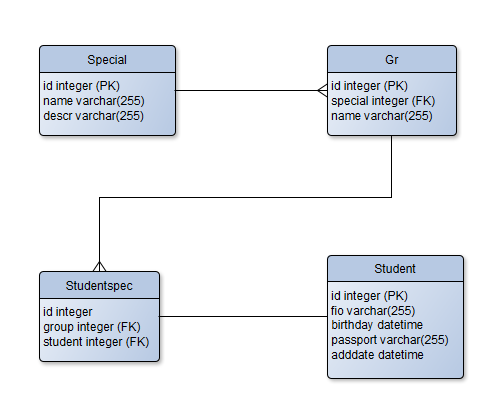


Рис. 2.2 Er-диаграмма базы данных.

Special – таблица со специальностями.

Gr – таблица с группами относящимся к специальностям. К каждой специальности может принадлежать несколько групп.

Student – таблица со студентами.

Studentspec – таблица устанавливает однозначное соответствие группа-студент. К каждой группе может принадлежать несколько студентов, но студент может принадлежать только к одной группе в один момент времени.

## UML-диаграмма классов

Диаграммы классов используются при моделировании программных систем наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

Класс – это основной строительный блок программных систем. Это понятие присутствует и в ОО языках программирования, то есть между классами UML и программными классами есть соответствие, являющееся основой для автоматической генерации программных кодов или для выполнения реинжиниринга. Каждый класс имеет название, атрибуты и операции. Класс на диаграмме показывается в виде прямоугольника, разделенного на 3 области. В верхней содержится название класса, в средней – описание атрибутов (свойств), в нижней – названия операций – услуг, предоставляемых объектами этого класса.

Атрибуты класса определяют состав и структуру данных, хранимых в объектах этого класса. Каждый атрибут имеет имя и тип, определяющий, какие данные он представляет. При реализации объекта в программном коде для атрибутов будет выделена память, необходимая для хранения всех атрибутов, и каждый атрибут будет иметь конкретное значение в любой момент времени работы программы. Объектов одного класса в программе может быть сколь угодно много, все они имеют одинаковый набор атрибутов, описанный в классе, но значения атрибутов у каждого объекта свои и могут изменяться в ходе выполнения программы.

Для каждого атрибута класса можно задать видимость (visibility). Эта характеристика показывает, доступен ли атрибут для других классов. В UML определены следующие уровни видимости атрибутов:

1. Открытый (public) – атрибут виден для любого другого класса (объекта);
2. Защищенный (protected) – атрибут виден для потомков данного класса;
3. Закрытый (private) – атрибут не виден внешними классами (объектами) и может использоваться только объектом, его содержащим.

Диаграмма классов представлена на Рис 2.3.



Рис. 2.3 UML диаграмма классов.

# Глава 3. Разработка

## 3.1 Принципы экстремального программирования

Название методологии исходит из идеи применить полезные традиционные методы и практики разработки программного обеспечения, подняв их на новый «экстремальный» уровень[4].

При разработке использовались принцип «Игра в планирование», «Непрерывная интеграция» и «Частые небольшие релизы».

Основная цель игры в планирование – быстро сформировать приблизительный план работы и постоянно обновлять его по мере того, как условия задачи становятся всё более чёткими. Артефактами игры в планирование является набор бумажных карточек, на которых записаны пожелания заказчика (customer stories), и приблизительный план работы по выпуску следующих одной или нескольких небольших версий продукта. Критическим фактором, благодаря которому такой стиль планирования оказывается эффективным, является то, что в данном случае заказчик отвечает за принятие бизнес-решений, а команда разработчиков отвечает за принятие технических решений.

Непрерывная интеграция – если выполнять интеграцию разрабатываемой системы достаточно часто, то можно избежать большей части связанных с этим проблем. В традиционных методиках интеграция, как правило, выполняется в самом конце работы над продуктом, когда считается, что все составные части разрабатываемой системы полностью готовы.

Частые небольшие релизы – версии (releases) продукта должны поступать в эксплуатацию как можно чаще. Работа над каждой версией должна занимать как можно меньше времени. При этом каждая версия должна быть достаточно осмысленной с точки зрения полезности для бизнеса.

При разработке приложения был составлен приблизительный план работ, который в процессе дополнялся и изменялся. После создания каждой небольшой функции она интегрировалась в приложение и тестировалась. Тем самым выполнялся и принцип частых небольших релизов, поскольку после ввода каждой новой функции приложение было работоспособным и его можно было использовать.

## 3.2 Используемые сторонние библиотеки для разработки

Для реализации базы данных в приложении используется SQLite. Это компактная встраиваемая реляционная база данных. Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа.

Хотя сама библиотека написана на языке C, однако существует множество оболочек, позволяющих использовать ее в связке с другими языками программирования, включая C#. Наиболее актуальным решением для использования функционала SQLite в связке с C# является провайдер ADO.NET, разрабатываемый непосредственно командой проекта SQLite.

Для добавления библиотеки к проекту был использован менеджер пакетов NuGet для Visual Studio, для установки использовалась команда:

PM> Install-Package System.Data.SQLite

Для сохранения отчета в формате pdf использовалась библиотека PDFsharp [1]. Для ее добавления в проект необходимо ее скачать архив с сайта и добавить ссылки на библиотеки (Рис 3.1). Библиотека предназначена для платформы net и есть две версии: для wpf и gpx, поскольку разрабатываемая программа использует windows forms, то выбран вариант gpx.

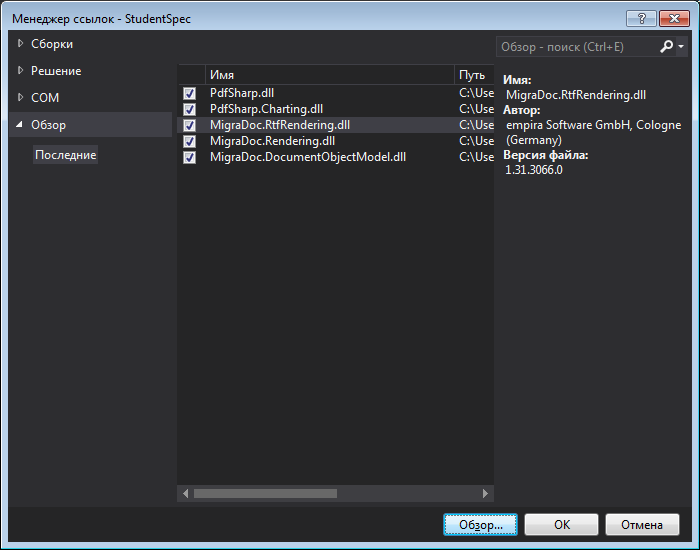
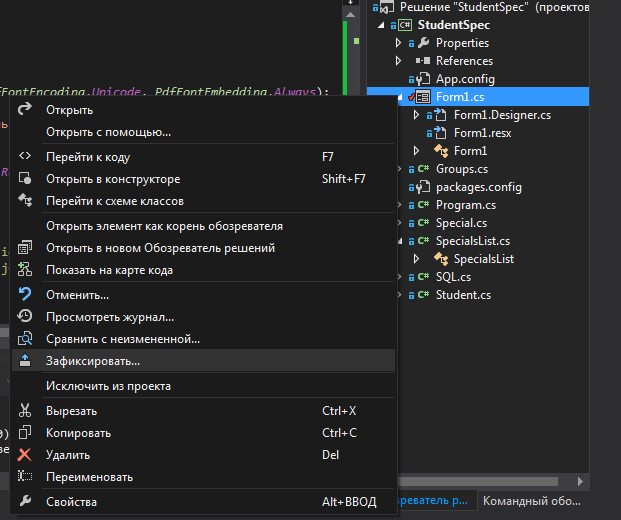


Рис. 3.1

## Использование системы контроля версий

При разработке приложения использовалась система контроля версий Git. Система контроля версий (СКВ) – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

Git предлагает использование как командной консоли, так и графической оболочки, кроме того при работе в Visual Studio можно использовать инструменты самой среды для фиксации изменений (Рис. 3.2). При наличии изменений в каких-либо файлах, в окне обозревателя решений рядом с файлом появляется красная галочка. При вызове контекстного меню, необходимо выбрать пункт Зафиксировать.

Рис.3.2 Изменения в файле

После этого будет открыто окно, в котором отобразятся все файлы, которые были выбраны для фиксации и будет предложено ввести комментарий к комиту (Рис.3.3).

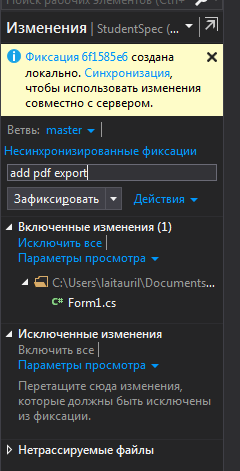


Рис.3.3 Фиксация изменений

Для работы с оболочками Git, необходимо перейти в директорию проекта и с помощью контекстного меню запустить консольную или оконную программу (Рис.3.4).

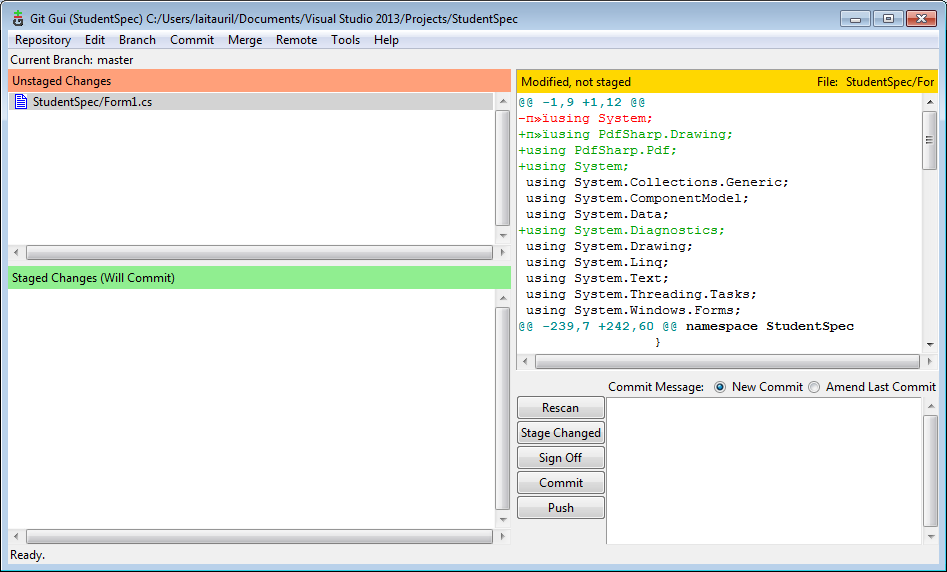


Рис.3.4 Просмотр измененного файла в Git GUI.

В ней также можно посмотреть все измененные файлы и зафиксировать изменения. Также в ней можно отобразить дерево фиксаций и перейти к нужной при необходимости (Рис.3.5).

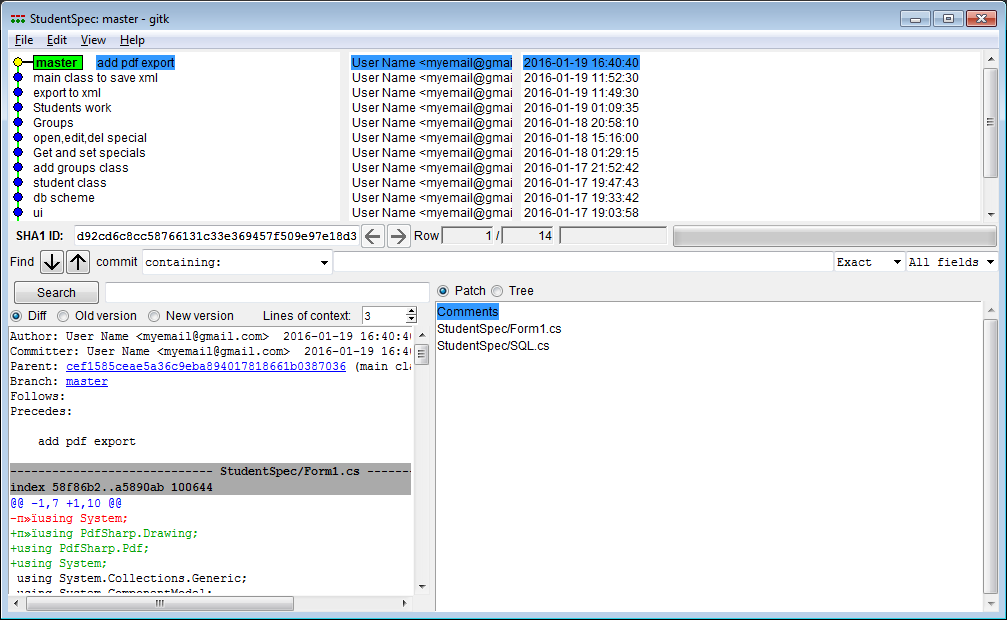


Рис.3.5 Просмотр дерева изменений

## Начало разработки

В начале нужно было создать главный файл, в котором будет брать начало наша программа. После главного файла, в нашем случае формы, было необходимо создать меню и интерфейс для нее. Следующим шагом было написание функции. После этого была написана страница отображения заказчиков, заказов и продавцов. И в самом конце были написаны и приведены в презентабельный вид такие аспекты программы как защита от нарушения правильной работы алгоритма программа (т.е блокираторы), изменение размеров окон и подобное.

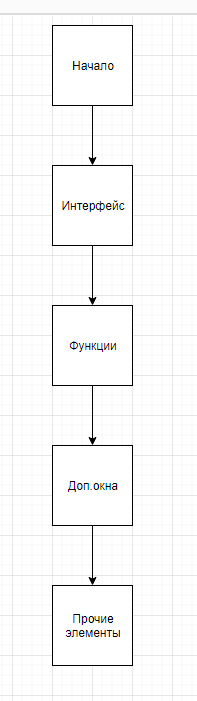


Рис.3.6 Дерево приоритетов при разработке

## Список файлов

Ниже представлен список файлов программы:

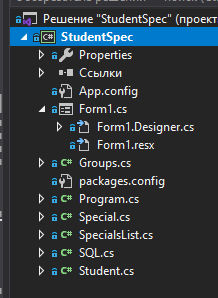


Рис.3.7 Список файлов.

SQL – класс реализует логику взаимодействия с SQLite.

Students – класс для хранения информации о студентах. Каждый экземпляр класса содержит информацию о одном конкретном студенте.

Groups – класс для хранения информации о группах. Также этот класс содержит список объектов класса Student, т.е. студентов, обучающихся в этой группе.

Specials – список специальностей, в качестве одного из полей содержит список объектов класса Groups.

SpecialsList – класс содержит список специальностей, т.е. объектов класса Specials, введен для сериализации и сохранения всех специальностей в xml.

Form1 – класс реализующий интерфейс программы, взаимодействует со всеми описанными выше классами.

Рассмотрим один из классов программы к примеру, SQL этот класс, класс реализует логику взаимодействия с SQLite, а также содержит строку подключения.  
**Листинг 1.** Класс SQL.

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StudentSpec

{

class SQL

{

//строка подключения

string sqlConnection = "Data Source=database.db;Version=3;New=True;Compress=True;datetimeformat=CurrentCulture";

//подключение

SQLiteConnection sqlite\_conn;

public SQL()

{

// создаем подключение

sqlite\_conn = new SQLiteConnection(sqlConnection);

// открываем его

sqlite\_conn.Open();

}

public void close()

{

//закрытие соединения

sqlite\_conn.Close();

}

//перевод даты в строку, для записи в бд

private string DateTimeSQLite(DateTime datetime)

{

string dateTimeFormat = "{0}-{1}-{2} {3}:{4}:{5}.{6}";

return string.Format(dateTimeFormat, datetime.Year, datetime.Month, datetime.Day, datetime.Hour, datetime.Minute, datetime.Second, datetime.Millisecond);

}

public void editStudent(int id, string fio, DateTime birthday, string passport)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE student SET fio='" + fio + "', birthday='" + DateTimeSQLite(birthday) +

"', passport='" + passport + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public int addStudent(string fio, DateTime birthday, string passport)

{

SQLiteDataReader sqlite\_datareader;

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO student (id, fio, birthday, passport, adddate) VALUES (null, '" + fio + "', '" + DateTimeSQLite(birthday) +

"', '" + passport + "', '" + DateTimeSQLite(DateTime.Now) + "');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "SELECT id FROM student ORDER BY id desc LIMIT 1";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

int id = 0;

// если есть данные

if (sqlite\_datareader.Read())

{

id = (int)(Int64)sqlite\_datareader["id"];

}

return id;

}

public void addSpecial(string name, string descr)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO special (id, name, descr) VALUES (null, '"+name+"', '"+descr+"');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addGroup(string name, int special)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO gr (id, name, special) VALUES (null, '" + name + "', " + special + ");";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addStudentSpec(int student, int group)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO studspec (id, student, gr) VALUES (null, " + student + ", " + group + ");";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void setStudentSpec(int student, int group)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE studspec SET gr=" + group + " WHERE student=" + student;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addHistory(string student, string grs, string grd)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO history (id, student, grs, grd) VALUES (null, '" + student + "', '" + grs + "', '"+grd+"');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void editGr(int id, string name)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE gr SET name='" + name + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void editSpecial(int id, string name, string descr)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE special SET name='" + name + "', descr='" + descr + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteGr(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "DELETE FROM gr WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteStudent(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

//удаляем студента

sqlite\_cmd.CommandText = "DELETE FROM student WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

//удаляем соответствия студент-группа

sqlite\_cmd.CommandText = "DELETE FROM studspec WHERE student=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteSpecial(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "DELETE FROM special WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

//создание базы данных

public void createScheme()

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists special (id integer primary key, name varchar(100), descr varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists student (id integer primary key, fio varchar(100), birthday datetime, passport varchar(100), adddate datetime);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists gr (id integer primary key, name varchar(100), special integer);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists studspec (id integer primary key, student integer, gr integer);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists history (id integer primary key, student varchar(100), grs varchar(100), grd varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

}

public int getStudentCount(int gr)

{

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT count(id) as cnt FROM studspec WHERE gr=" + gr;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

int sum = 0;

// если есть данные для чтения

if (sqlite\_datareader.Read())

{

sum = (int)(Int64)sqlite\_datareader["cnt"];

}

return sum;

}

public List<Student> getListStudents(int gr, int special)

{

List<Student> res = new List<Student>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT student.id,fio,birthday,passport,adddate FROM student,studspec WHERE studspec.student=student.id AND studspec.gr=" + gr;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

// пока есть данные для чтения

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

DateTime bdate = (DateTime)sqlite\_datareader["birthday"];

DateTime adate = (DateTime)sqlite\_datareader["adddate"];

res.Add(new Student

{

id = (int)\_id,

fio = (string)sqlite\_datareader["fio"],

pasport = (string)sqlite\_datareader["passport"],

gr = gr,

special = special,

birthday = bdate,

createDate = adate

});

}

return res;

}

public List<Groups> getListGr(int special)

{

List<Groups> res = new List<Groups>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT id,name FROM gr WHERE special=" + special;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

string \_name = (string)sqlite\_datareader["name"];

res.Add(new Groups

{

id = (int)\_id,

name = \_name,

special = special,

students = getStudentCount((int)\_id)

});

}

return res;

}

public List<Special> getListSpecial()

{

List<Special> res = new List<Special>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT id,name,descr FROM special";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

string \_name = (string)sqlite\_datareader["name"];

string \_descr = (string)sqlite\_datareader["descr"];

res.Add(new Special{id = (int)\_id,

name = \_name,

descr = \_descr

});

}

return res;

}

public void test()

{

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "CREATE TABLE test (id integer primary key, text varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "INSERT INTO test (id, text) VALUES (1, 'Test Text 1');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "INSERT INTO test (id, text) VALUES (2, 'Test Text 2');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "SELECT \* FROM test";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

System.Console.WriteLine(sqlite\_datareader["text"]);

}

}

}

В данном классе были написаны методы, которые выполняют определенные операции рассмотрим один из них:

1. addStudent – данный метод служит для добавления студента. Так же метод принимает данные такие как «string fio, DateTime birthday, string passport».

Листинг 2. Метод addStudent.

public int addStudent(string fio, DateTime birthday, string passport)

{

SQLiteDataReader sqlite\_datareader;

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO student (id, fio, birthday, passport, adddate) VALUES (null, '" + fio + "', '" + DateTimeSQLite(birthday) +

"', '" + passport + "', '" + DateTimeSQLite(DateTime.Now) + "');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "SELECT id FROM student ORDER BY id desc LIMIT 1";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

int id = 0;

// если есть данные

if (sqlite\_datareader.Read())

{

id = (int)(Int64)sqlite\_datareader["id"];

}

return id;

}

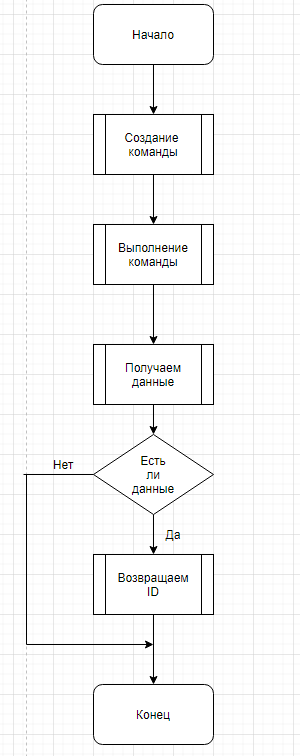


Рис.3.6 Блок схема метода addStudent.

# Глава 4. Эксплуатация

## 4.1 Требуемое программное и аппаратное обеспечение для работы вашего приложения.

Для работы приложения требуется программное обеспечение:

1. Microsoft Framework 4.0,
2. Microsoft Windows XP или выше.

Аппаратное обеспечение:

1. Процессор: Intel® Pentium 4 1.5GHz и выше
2. Видеокарта: любая
3. Оперативная память: не менее 512 Мбайт
4. Пространство на жестком диске: 50 Мбайт
5. Монитор: SVGA или выше
6. Наличие клавиатуры, мыши

## 4.2 Описание установки разработанного приложения.

Для установки приложения необходимо извлечь файлы из архива. Во время первого запуска будет создана база данных. База данных будет хранится в той же директории, что и исполняемый файл, и будет использоваться при последующих запусках.

## 4.3 Описание для пользователя

Для запуска приложения необходимо запустить исполняемый файл Studentpec.exe (Рис.4.1).

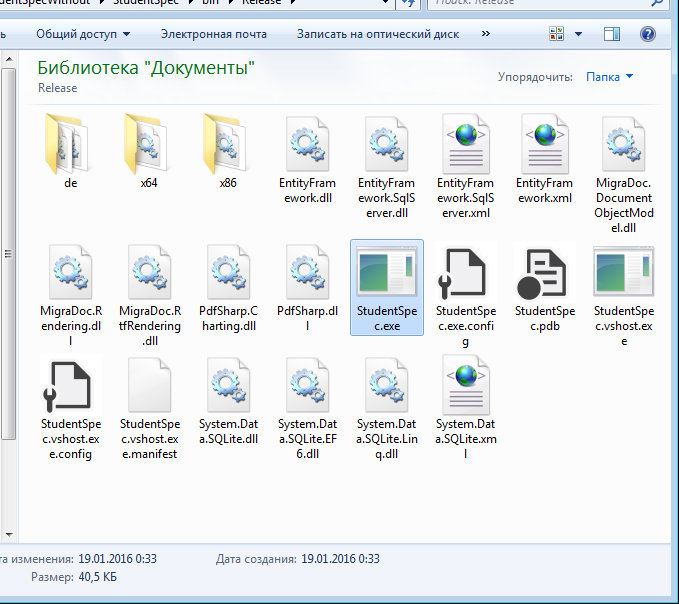


Рис.4.1 Запуск приложения

После запуска откроется окно программы (Рис.4.2) с возможностью работы со специальностями, а также возможностью представления информации в pdf отчете, графике или xml. При первом запуске, информация в таблицах будет отсутствовать.

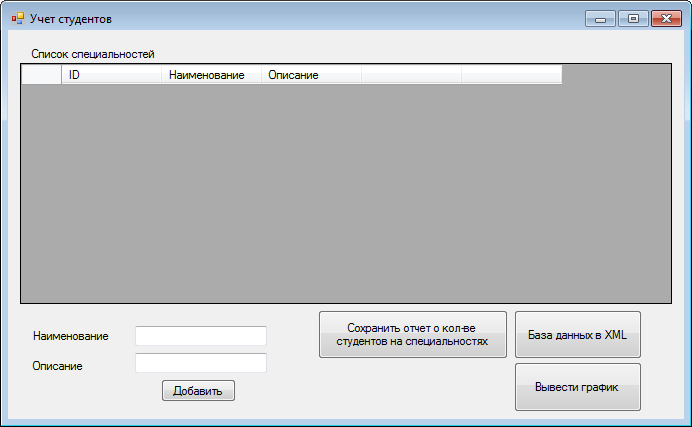


Рис.4.2 Главное окно программы

Для добавления специальности, необходимо заполнить соответствующие поля и нажать кнопку Добавить. Добавленную специальность можно будет отредактировать, изменив данные в таблице. При нажатии кнопки Открыть (Рис.4.3) окно программы перейдет в состояние работы с группами (Рис.4.4).

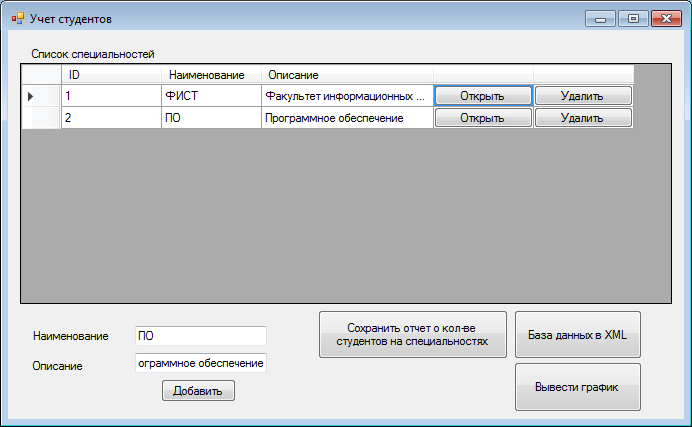


Рис.4.3 Окно с добавленными специальностями

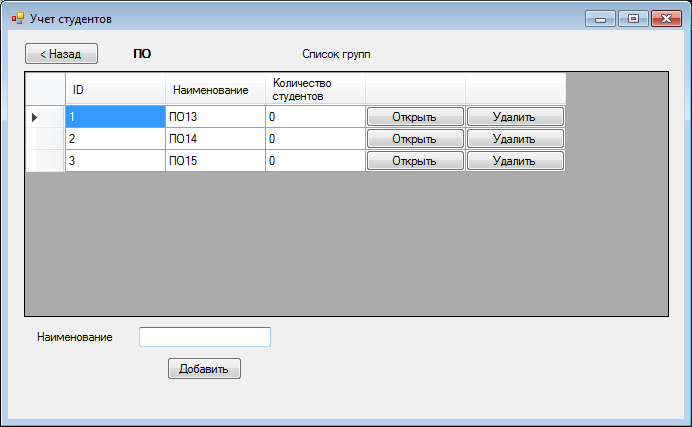


Рис.4.4 Окно со списком групп.

Работа со списком групп и списком студентов (Рис.4.5) производится аналогичным образом, как и со списком специальностей.

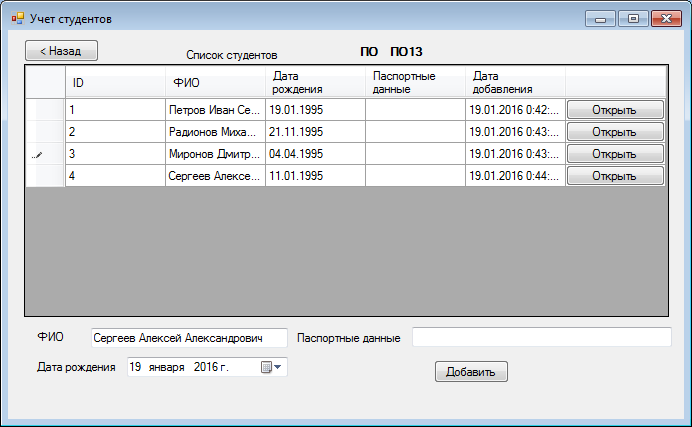


Рис.4.5 Список студентов

Для исключения или перевода студента, необходимо нажать кнопку Открыть. И после этого выбрать нужную специальность (Рис.4.6).

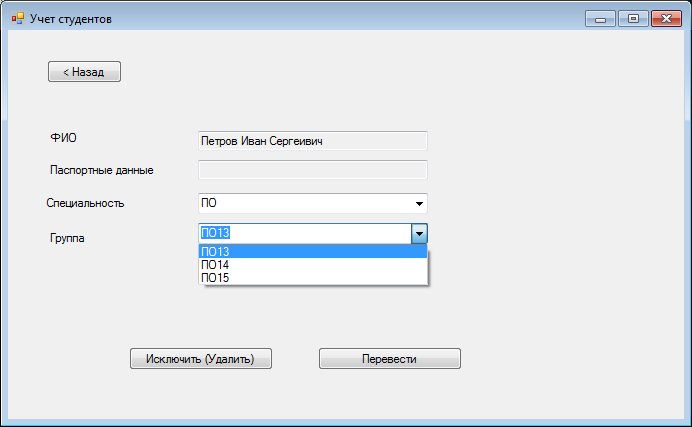


Рис.4.6 Перевод в другую группу.

Для возврата к списку специальностей используется кнопка Назад. При сохранении отчета или базы данных в xml, вызывается диалог как на Рис.4.7, в котором пользователь может выбрать нужное место.

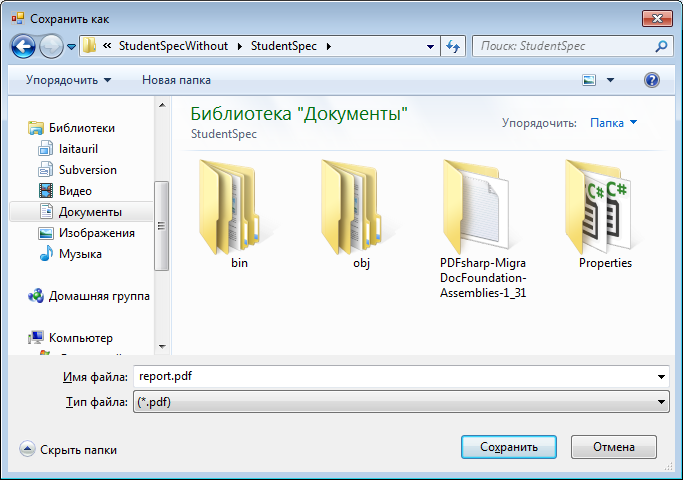


Рис.4.7 Диалог сохранения отчета.

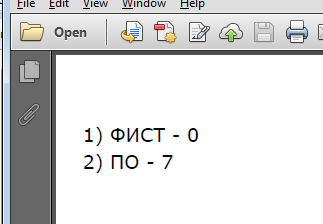


Рис.4.8 Сохраненный отчет в pdf

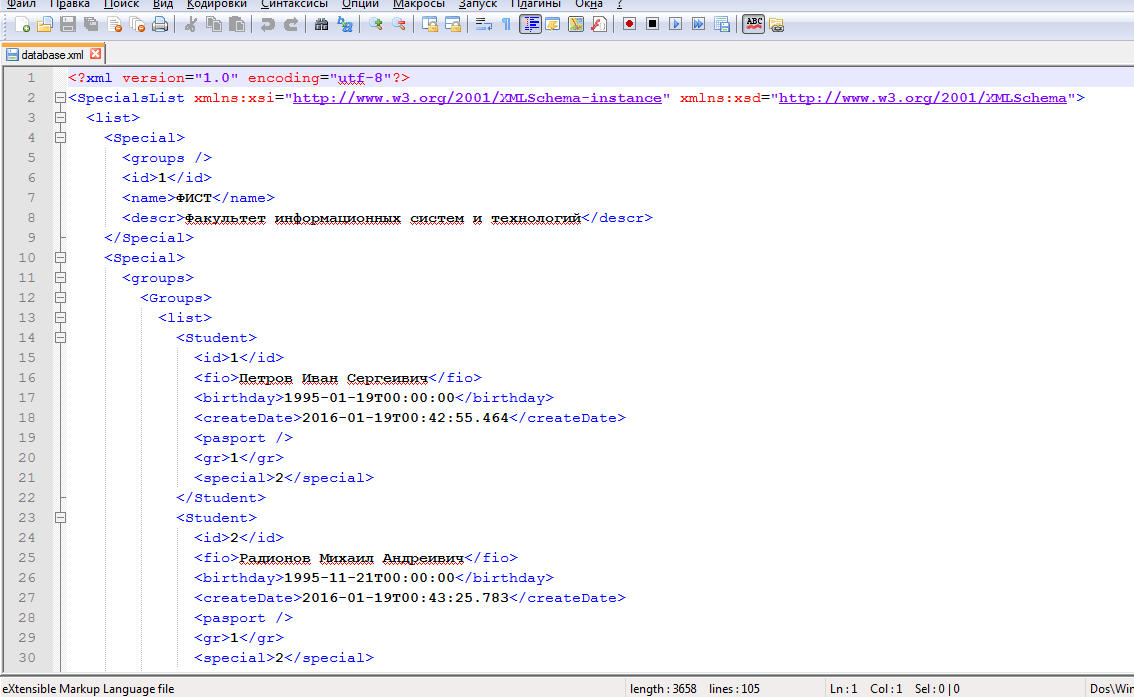


Рис.4.9 Сохраненная база данных в XML.

График выводится на форме, по горизонтальной оси показано количество человек, по вертикальной идут группы (Рис.4.10).

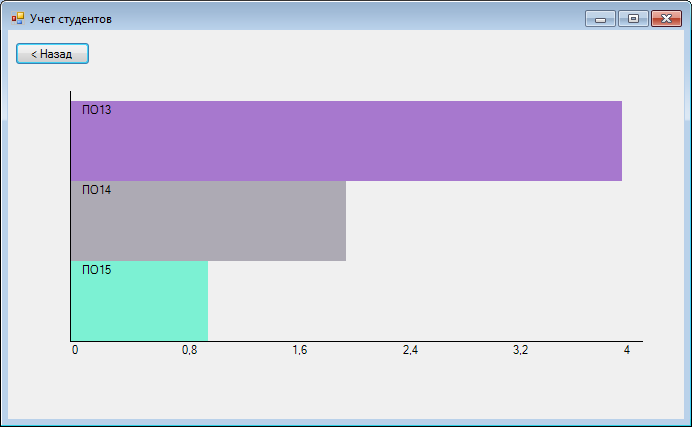


Рис.4.10 График количества людей в группах

# Заключение.

В курсовой работе было разработано приложение для учета студентов по группам и специальностям. Приложение хранит свои данный в SQLite, что не требует установки дополнительного программного обеспечения и наличия выхода в сеть.

Приложение умеет формировать отчет в формате pdf о количестве студентов на специальностях, выводить график о количестве студентов в группах и экспортировать всю базу данных в XML.

Приложение можно запускать на любом компьютере с установленной версией Framework 4.0.

# Библиографический список.

1. <http://habrahabr.ru/post/47940/> Use-case диаграмма
2. <https://ru.wikipedia.org/wiki/ER-модель_данных>
3. <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3> диаграмма классов
4. https://ru.wikipedia.org/wiki/Экстремальное\_программирование
5. <http://www.pdfsharp.net/wiki/?AspxAutoDetectCookieSupport=1> PDFsharp.
6. В.В. Фаронов “C 6: Учебный Курс”, Москва, Knowledge, 2001
7. М.В. Сухарев “Основы C#”, издательство “Наука и техника”, 2003
8. Г.В. Галисеев, “Компоненты в Delphi 6: профессиональная работа”, издательство “Диалектика”, 2004
9. В.И. Ключко “Методическое указание к выполнению курсовой работы”, Краснодар, Издательство КубГТУ, 1997
10. М.П. Малыхина “Базы данных: основы, проектирование, использование”
11. Т.М. Карпова “Базы данных: модели, разработка, реализация”, издательство “Питер”, 2001
12. Кандзюба С. П. “Delphi 6/7. Базы данных и приложения. Лекции и упражнения”. – СПб.: ООО «ДиаСофтЮП», 2002. – 576.

# Приложение

**Листинг 1**. Главное окно программы Form1.cs С#.

using PdfSharp.Drawing;

using PdfSharp.Pdf;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace StudentSpec

{

public partial class Form1 : Form

{

SQL sql; //класс работы с sqlite

int currentSpecial = 0; //id выбранной пользователем специальности

int currentGr = 0; //id выбранной группы

int currentStudent = 0; //id выбранного студента

List<Special> listSpecials; //список специальснотей

List<Groups> listGr; //список групп для выбранной специальности

int currentSpecialInd = 0; //номер выбранной специальности в списке

int currentGrInd = 0; //номер выбранной группы в списке

List<Groups> listGrTemp; //список групп, используется для перевода студента на др. специальность/группу

Dictionary<string, int> graphic = null; //для вывода графика, хранит пары: наименование группы - количество студентов

List<Color> colors; //список цветов для отображения колонок в графике

public Form1()

{

InitializeComponent();

//подключаемся к бд

sql = new SQL();

sql.createScheme();

//загружаем список специальснотей

updateSpecial();

}

private void buttonAddSpec\_Click(object sender, EventArgs e)

{

sql.addSpecial(textBoxSpecName.Text, textBoxSpecDescr.Text);

updateSpecial();

}

public void updateSpecial()

{

listSpecials = sql.getListSpecial();

dataGridViewSpecial.Rows.Clear();

dataGridViewSpecial.RowCount = listSpecials.Count;

comboBoxSpecial.Items.Clear();

for (int i = 0; i < listSpecials.Count; i++)

{

dataGridViewSpecial.Rows[i].Cells[0].Value = listSpecials[i].id;

dataGridViewSpecial.Rows[i].Cells[1].Value = listSpecials[i].name;

dataGridViewSpecial.Rows[i].Cells[2].Value = listSpecials[i].descr;

dataGridViewSpecial.Rows[i].Cells[3].Value = "Открыть";

dataGridViewSpecial.Rows[i].Cells[4].Value = "Удалить";

comboBoxSpecial.Items.Add(listSpecials[i].name);

}

}

public void updateGr(int special)

{

listGr = sql.getListGr(special);

dataGridViewGr.Rows.Clear();

dataGridViewGr.RowCount = listGr.Count;

comboBoxGr.Items.Clear();

for (int i = 0; i < listGr.Count; i++)

{

dataGridViewGr.Rows[i].Cells[0].Value = listGr[i].id;

dataGridViewGr.Rows[i].Cells[1].Value = listGr[i].name;

dataGridViewGr.Rows[i].Cells[2].Value = listGr[i].students;

dataGridViewGr.Rows[i].Cells[3].Value = "Открыть";

dataGridViewGr.Rows[i].Cells[4].Value = "Удалить";

comboBoxGr.Items.Add(listGr[i].name);

}

}

//перевод даты в строку для вывода пользователю

private string dateToStr(DateTime dt)

{

string res;

if (dt.Day < 10)

res = "0" + dt.Day + ".";

else

res = dt.Day + ".";

if (dt.Month < 10)

res += "0" + dt.Month + ".";

else

res += dt.Month + ".";

res += dt.Year;

return res;

}

public void updateStudents(int gr)

{

List<Student> list = sql.getListStudents(gr, currentSpecial);

dataGridViewStudents.Rows.Clear();

dataGridViewStudents.RowCount = list.Count;

for (int i = 0; i < list.Count; i++)

{

dataGridViewStudents.Rows[i].Cells[0].Value = list[i].id;

dataGridViewStudents.Rows[i].Cells[1].Value = list[i].fio;

dataGridViewStudents.Rows[i].Cells[2].Value = dateToStr(list[i].birthday);

dataGridViewStudents.Rows[i].Cells[3].Value = list[i].pasport;

dataGridViewStudents.Rows[i].Cells[4].Value = list[i].createDate;

dataGridViewStudents.Rows[i].Cells[5].Value = "Открыть";

}

}

private void dataGridViewSpecial\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

sql.editSpecial((int)dataGridViewSpecial.Rows[e.RowIndex].Cells[0].Value,

(string)dataGridViewSpecial.Rows[e.RowIndex].Cells[1].Value,

(string)dataGridViewSpecial.Rows[e.RowIndex].Cells[2].Value);

}

//клик по кнопкам в таблице специальностей

private void dataGridViewSpecial\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.ColumnIndex==3)

{

//открыть

tabControl1.SelectedIndex = 1;

labelSpecial.Text = (string)dataGridViewSpecial.Rows[e.RowIndex].Cells[1].Value;

currentSpecial = (int)dataGridViewSpecial.Rows[e.RowIndex].Cells[0].Value;

currentSpecialInd = e.RowIndex;

updateGr(currentSpecial);

comboBoxSpecial.SelectedIndex = e.RowIndex;

}

else if (e.ColumnIndex == 4)

{

//удалить

sql.deleteSpecial((int)dataGridViewSpecial.Rows[e.RowIndex].Cells[0].Value);

updateGr(currentSpecial);

}

}

private void button2\_Click(object sender, EventArgs e)

{

tabControl1.SelectedIndex = 0;

}

private void buttonAddGroup\_Click(object sender, EventArgs e)

{

sql.addGroup(textBoxGroup.Text, currentSpecial);

updateGr(currentSpecial);

}

private void dataGridViewGr\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

sql.editGr((int)dataGridViewGr.Rows[e.RowIndex].Cells[0].Value,

(string)dataGridViewGr.Rows[e.RowIndex].Cells[1].Value);

}

private void buttonAddStudent\_Click(object sender, EventArgs e)

{

int id = sql.addStudent(textBoxFIO.Text, dateTimePickerBirthday.Value, textBoxPasport.Text);

sql.addStudentSpec(id, currentGr);

updateStudents(currentGr);

}

private void dataGridViewGr\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.ColumnIndex == 3)

{

//открыть

tabControl1.SelectedIndex = 2;

labelSpecGr.Text = labelSpecial.Text + " " + (string)dataGridViewGr.Rows[e.RowIndex].Cells[1].Value;

currentGr = (int)dataGridViewGr.Rows[e.RowIndex].Cells[0].Value;

currentGrInd = e.RowIndex;

updateStudents(currentGr);

comboBoxGr.SelectedIndex = e.RowIndex;

}

else if (e.ColumnIndex == 4)

{

//удалить

sql.deleteGr((int)dataGridViewGr.Rows[e.RowIndex].Cells[0].Value);

updateStudents(currentGr);

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

sql.close();

}

private void button1\_Click(object sender, EventArgs e)

{

tabControl1.SelectedIndex = 1;

updateGr(currentGr);

}

private void dataGridViewStudents\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

DateTime dt;

if (dataGridViewStudents.Rows[e.RowIndex].Cells[2].Value is DateTime)

dt = (DateTime)dataGridViewStudents.Rows[e.RowIndex].Cells[2].Value;

else

{

string str = (string)dataGridViewStudents.Rows[e.RowIndex].Cells[2].Value;

dt = DateTime.Parse(str);

}

sql.editStudent((int)dataGridViewStudents.Rows[e.RowIndex].Cells[0].Value,

(string)dataGridViewStudents.Rows[e.RowIndex].Cells[1].Value,

dt,

(string)dataGridViewStudents.Rows[e.RowIndex].Cells[3].Value);

}

private void dataGridViewStudents\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.ColumnIndex == 5)

{

//открыть

tabControl1.SelectedIndex = 3;

currentStudent = (int)dataGridViewStudents.Rows[e.RowIndex].Cells[0].Value;

textBoxDetailFIO.Text = (string)dataGridViewStudents.Rows[e.RowIndex].Cells[1].Value;

textBoxDetailPassport.Text = (string)dataGridViewStudents.Rows[e.RowIndex].Cells[3].Value;

updateStudentInfo();

}

}

public void updateStudentInfo()

{

comboBoxGr.Items.Clear();

for (int i = 0; i < listGr.Count; i++)

{

comboBoxGr.Items.Add(listGr[i].name);

}

comboBoxGr.SelectedIndex = currentGrInd;

}

private void comboBoxSpecial\_SelectedIndexChanged(object sender, EventArgs e)

{

listGrTemp = sql.getListGr(listSpecials[comboBoxSpecial.SelectedIndex].id);

comboBoxGr.Items.Clear();

for (int i = 0; i < listGrTemp.Count; i++)

{

comboBoxGr.Items.Add(listGrTemp[i].name);

}

}

private void button3\_Click(object sender, EventArgs e)

{

tabControl1.SelectedIndex = 2;

updateStudents(currentGr);

}

private void button4\_Click(object sender, EventArgs e)

{

sql.deleteStudent(currentStudent);

button3\_Click(this, null);

}

private void buttonStudentSpecSet\_Click(object sender, EventArgs e)

{

sql.setStudentSpec(currentStudent, listGrTemp[comboBoxGr.SelectedIndex].id);

}

private void button5\_Click(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog1 = new SaveFileDialog();

string ReadResFilePlace = null;

//разрешение файлов

saveFileDialog1.Filter = "(\*.xml)|\*.xml|All files (\*.\*)|\*.\*";

//имя файла по умолчнию

saveFileDialog1.FileName = "database";

saveFileDialog1.DefaultExt = "xml";

//показываем диалог сохранения

DialogResult drs = saveFileDialog1.ShowDialog();

ReadResFilePlace = saveFileDialog1.FileName;

if (ReadResFilePlace != String.Empty)

{

SpecialsList sl = new SpecialsList();

sl.list = sql.getListSpecial();

for (int i = 0; i < sl.list.Count; i++)

{

sl.list[i].groups = sql.getListGr(sl.list[i].id);

for (int j = 0; j < sl.list[i].groups.Count; j++)

{

sl.list[i].groups[j].list = sql.getListStudents(sl.list[i].groups[j].id, sl.list[i].id);

}

}

SpecialsList.SerializeObject(ReadResFilePlace, sl);

}

}

private void button4\_Click\_1(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog1 = new SaveFileDialog();

string ReadResFilePlace = null;

//разрешение файлов

saveFileDialog1.Filter = "(\*.pdf)|\*.pdf|All files (\*.\*)|\*.\*";

//имя файла по умолчнию

saveFileDialog1.FileName = "report";

saveFileDialog1.DefaultExt = "pdf";

//показываем диалог сохранения

DialogResult drs = saveFileDialog1.ShowDialog();

ReadResFilePlace = saveFileDialog1.FileName;

if (ReadResFilePlace != String.Empty)

{

SpecialsList sl = new SpecialsList();

sl.list = sql.getListSpecial();

string text = "";

PdfDocument pdf = new PdfDocument();

XPdfFontOptions options = new XPdfFontOptions(PdfFontEncoding.Unicode, PdfFontEmbedding.Always);

pdf.Info.Title = "Количество студентов на специальностях";

PdfPage pdfPage = pdf.AddPage();

XGraphics graph = XGraphics.FromPdfPage(pdfPage);

XFont font = new XFont("Verdana", 14, XFontStyle.Regular, options);

for (int i = 0; i < sl.list.Count; i++)

{

int num = 0;

sl.list[i].groups = sql.getListGr(sl.list[i].id);

for (int j = 0; j < sl.list[i].groups.Count; j++)

{

num += sl.list[i].groups[j].students;

}

graph.DrawString( " " + (i+1) + ") " +

sl.list[i].name + " - " + num, font,

XBrushes.Black,

new XRect(0, 50 + i\*20, pdfPage.Width.Point, pdfPage.Height.Point),

XStringFormats.TopLeft);

}

pdf.Save(ReadResFilePlace);

Process.Start(ReadResFilePlace);

}

}

private void button7\_Click(object sender, EventArgs e)

{

tabControl1.SelectedIndex = 0;

}

private void button6\_Click(object sender, EventArgs e)

{

graphic = new Dictionary<string, int>();

SpecialsList sl = new SpecialsList();

sl.list = sql.getListSpecial();

colors = new List<Color>(); //создаем список цетов

Random r = new Random();

for (int i = 0; i < sl.list.Count; i++)

{

sl.list[i].groups = sql.getListGr(sl.list[i].id);

for (int j = 0; j < sl.list[i].groups.Count; j++)

{

//вносим значение

graphic.Add(sl.list[i].groups[j].name, sl.list[i].groups[j].students);

//цвета определяются случайно, но выбираются светлые тона, чтобы были видны надписи на них

colors.Add(Color.FromArgb(100 + r.Next(155), 100 + r.Next(155), 100 + r.Next(155)));

}

}

//переходим на вкладку с графиком

tabControl1.SelectedIndex = 4;

//вызываем перерисовку формы

Invalidate();

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

//если нет данных, выходим

if (graphic == null)

return;

//ищем максимальное количесто студентов

int maxStudent = 0;

foreach (KeyValuePair<string, int> pair in graphic)

{

if (pair.Value > maxStudent)

maxStudent = pair.Value;

}

//высота прямоугольников

int rectHeight = (panel1.Height - 100) / graphic.Count;

//коэфициент длины прямоугольников

int kWidth = (panel1.Width - 100)/maxStudent;

Graphics gr = e.Graphics;

int i = 0;

foreach (KeyValuePair<string, int> pair in graphic)

{

SolidBrush myBrush = new SolidBrush(colors[i]);

gr.FillRectangle(myBrush, 50, 30 + i \* rectHeight, kWidth \* pair.Value, rectHeight);

gr.DrawString(pair.Key, Font, Brushes.Black, new PointF(60, 32 + i \* rectHeight));

i++;

}

//выводим оси

gr.DrawLine(Pens.Black, 50, 20, 50, 30 + i \* rectHeight);

gr.DrawLine(Pens.Black, 50, 30 + i \* rectHeight, panel1.Width-30, 30 + i \* rectHeight);

//выводим цифры

for (int j=0; j<=5; j++)

{

gr.DrawString((j \* maxStudent/5.0f) + "", Font, Brushes.Black, new PointF(50+j \* (panel1.Width - 100) / 5, 32 + i \* rectHeight));

}

}

}

}

**Листинг 2.**

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StudentSpec

{

class SQL

{

//строка подключения

string sqlConnection = "Data Source=database.db;Version=3;New=True;Compress=True;datetimeformat=CurrentCulture";

//подключение

SQLiteConnection sqlite\_conn;

public SQL()

{

// создаем подключение

sqlite\_conn = new SQLiteConnection(sqlConnection);

// открываем его

sqlite\_conn.Open();

}

public void close()

{

//закрытие соединения

sqlite\_conn.Close();

}

//перевод даты в строку, для записи в бд

private string DateTimeSQLite(DateTime datetime)

{

string dateTimeFormat = "{0}-{1}-{2} {3}:{4}:{5}.{6}";

return string.Format(dateTimeFormat, datetime.Year, datetime.Month, datetime.Day, datetime.Hour, datetime.Minute, datetime.Second, datetime.Millisecond);

}

public void editStudent(int id, string fio, DateTime birthday, string passport)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE student SET fio='" + fio + "', birthday='" + DateTimeSQLite(birthday) +

"', passport='" + passport + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public int addStudent(string fio, DateTime birthday, string passport)

{

SQLiteDataReader sqlite\_datareader;

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO student (id, fio, birthday, passport, adddate) VALUES (null, '" + fio + "', '" + DateTimeSQLite(birthday) +

"', '" + passport + "', '" + DateTimeSQLite(DateTime.Now) + "');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "SELECT id FROM student ORDER BY id desc LIMIT 1";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

int id = 0;

// если есть данные

if (sqlite\_datareader.Read())

{

id = (int)(Int64)sqlite\_datareader["id"];

}

return id;

}

public void addSpecial(string name, string descr)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO special (id, name, descr) VALUES (null, '"+name+"', '"+descr+"');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addGroup(string name, int special)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO gr (id, name, special) VALUES (null, '" + name + "', " + special + ");";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addStudentSpec(int student, int group)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO studspec (id, student, gr) VALUES (null, " + student + ", " + group + ");";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void setStudentSpec(int student, int group)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE studspec SET gr=" + group + " WHERE student=" + student;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void addHistory(string student, string grs, string grd)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "INSERT INTO history (id, student, grs, grd) VALUES (null, '" + student + "', '" + grs + "', '"+grd+"');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void editGr(int id, string name)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE gr SET name='" + name + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void editSpecial(int id, string name, string descr)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "UPDATE special SET name='" + name + "', descr='" + descr + "' WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteGr(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "DELETE FROM gr WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteStudent(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

//удаляем студента

sqlite\_cmd.CommandText = "DELETE FROM student WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

//удаляем соответствия студент-группа

sqlite\_cmd.CommandText = "DELETE FROM studspec WHERE student=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

public void deleteSpecial(int id)

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "DELETE FROM special WHERE id=" + id;

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

}

//создание базы данных

public void createScheme()

{

SQLiteCommand sqlite\_cmd;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists special (id integer primary key, name varchar(100), descr varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists student (id integer primary key, fio varchar(100), birthday datetime, passport varchar(100), adddate datetime);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists gr (id integer primary key, name varchar(100), special integer);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists studspec (id integer primary key, student integer, gr integer);";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "CREATE TABLE if not exists history (id integer primary key, student varchar(100), grs varchar(100), grd varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

}

public int getStudentCount(int gr)

{

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT count(id) as cnt FROM studspec WHERE gr=" + gr;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

int sum = 0;

// если есть данные для чтения

if (sqlite\_datareader.Read())

{

sum = (int)(Int64)sqlite\_datareader["cnt"];

}

return sum;

}

public List<Student> getListStudents(int gr, int special)

{

List<Student> res = new List<Student>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT student.id,fio,birthday,passport,adddate FROM student,studspec WHERE studspec.student=student.id AND studspec.gr=" + gr;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

// пока есть данные для чтения

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

DateTime bdate = (DateTime)sqlite\_datareader["birthday"];

DateTime adate = (DateTime)sqlite\_datareader["adddate"];

res.Add(new Student

{

id = (int)\_id,

fio = (string)sqlite\_datareader["fio"],

pasport = (string)sqlite\_datareader["passport"],

gr = gr,

special = special,

birthday = bdate,

createDate = adate

});

}

return res;

}

public List<Groups> getListGr(int special)

{

List<Groups> res = new List<Groups>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT id,name FROM gr WHERE special=" + special;

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

string \_name = (string)sqlite\_datareader["name"];

res.Add(new Groups

{

id = (int)\_id,

name = \_name,

special = special,

students = getStudentCount((int)\_id)

});

}

return res;

}

public List<Special> getListSpecial()

{

List<Special> res = new List<Special>();

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "SELECT id,name,descr FROM special";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

Int64 \_id = (Int64)sqlite\_datareader["id"];

string \_name = (string)sqlite\_datareader["name"];

string \_descr = (string)sqlite\_datareader["descr"];

res.Add(new Special{id = (int)\_id,

name = \_name,

descr = \_descr

});

}

return res;

}

public void test()

{

SQLiteCommand sqlite\_cmd;

SQLiteDataReader sqlite\_datareader;

// создаем новую SQL комманду

sqlite\_cmd = sqlite\_conn.CreateCommand();

sqlite\_cmd.CommandText = "CREATE TABLE test (id integer primary key, text varchar(100));";

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "INSERT INTO test (id, text) VALUES (1, 'Test Text 1');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "INSERT INTO test (id, text) VALUES (2, 'Test Text 2');";

// выполняем комманду

sqlite\_cmd.ExecuteNonQuery();

sqlite\_cmd.CommandText = "SELECT \* FROM test";

// получаем данные для чтения

sqlite\_datareader = sqlite\_cmd.ExecuteReader();

while (sqlite\_datareader.Read())

{

System.Console.WriteLine(sqlite\_datareader["text"]);

}

}

}

}

Листинг 3. Класс Student.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

namespace StudentSpec

{

[Serializable]

public class Student

{

//номер студента в бд

public int id { get; set; }

public string fio { get; set; }

public DateTime birthday { get; set; }

//дата создания записи в бд

public DateTime createDate { get; set; }

//паспортные данные

public string pasport { get; set; }

//группа

public int gr { get; set; }

//специальносить

public int special { get; set; }

public Student()

{

}

//конструктор для сериализации

public Student(SerializationInfo sInfo, StreamingContext contextArg)

{

this.id = (int)sInfo.GetValue("id", typeof(int));

this.fio = (string)sInfo.GetValue("fio", typeof(string));

this.birthday = (DateTime)sInfo.GetValue("birthday", typeof(DateTime));

this.createDate = (DateTime)sInfo.GetValue("createDate", typeof(DateTime));

this.pasport = (string)sInfo.GetValue("pasport", typeof(string));

this.gr = (int)sInfo.GetValue("gr", typeof(int));

this.special = (int)sInfo.GetValue("special", typeof(int));

}

//получение данных для сериализации

public void GetObjectData(SerializationInfo sInfo, StreamingContext contextArg)

{

sInfo.AddValue("id", this.id);

sInfo.AddValue("fio", this.fio);

sInfo.AddValue("birthday", this.birthday);

sInfo.AddValue("createDate", this.createDate);

sInfo.AddValue("pasport", this.pasport);

sInfo.AddValue("gr", this.gr);

sInfo.AddValue("special", this.special);

}

}

}

Листинг 4. Класс SpecialList.cs  
using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace StudentSpec

{

[Serializable]

public class SpecialsList

{

//список специальностей

public List<Special> list;

public SpecialsList()

{

}

public SpecialsList(SerializationInfo sInfo, StreamingContext contextArg)

{

this.list = (List<Special>)sInfo.GetValue("list", typeof(List<Special>));

}

public void GetObjectData(SerializationInfo sInfo, StreamingContext contextArg)

{

sInfo.AddValue("list", this.list);

}

//сохранение в xml файл

public static void SerializeObject(string filename, SpecialsList objToSerialize)

{

XmlSerializer xmlSerializer = new XmlSerializer(typeof(SpecialsList));

//создаем текстовый поток на запись

TextWriter writer = new StreamWriter(filename);

//записываем

xmlSerializer.Serialize(writer, objToSerialize);

writer.Close();

}

}

}

Листинг 5. Класс Special.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

namespace StudentSpec

{

[Serializable]

public class Special

{

//номер специальности

public int id {get; set;}

//наименование специальности

public string name { get; set; }

//описание

public string descr { get; set; }

//список групп на этой специальности

public List<Groups> groups;

public Special()

{

}

public Special(SerializationInfo sInfo, StreamingContext contextArg)

{

this.id = (int)sInfo.GetValue("id", typeof(int));

this.name = (string)sInfo.GetValue("name", typeof(string));

this.descr = (string)sInfo.GetValue("descr", typeof(string));

this.groups = (List<Groups>)sInfo.GetValue("groups", typeof(List<Groups>));

}

public void GetObjectData(SerializationInfo sInfo, StreamingContext contextArg)

{

sInfo.AddValue("id", this.id);

sInfo.AddValue("name", this.name);

sInfo.AddValue("descr", this.descr);

sInfo.AddValue("groups", this.groups);

}

}

}

Листинг 6. Класс Group.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

namespace StudentSpec

{

[Serializable]

public class Groups

{

//номер группы

public int id {get; set;}

//номер специальности, к которой принадлежит группа

public int special { get; set; }

//наименование группы

public string name { get; set; }

//количество студентов в группе

public int students { get; set; }

//список студентов в группе

public List<Student> list;

public Groups()

{

}

public Groups(SerializationInfo sInfo, StreamingContext contextArg)

{

this.id = (int)sInfo.GetValue("id", typeof(int));

this.special = (int)sInfo.GetValue("special", typeof(int));

this.name = (string)sInfo.GetValue("name", typeof(string));

this.students = (int)sInfo.GetValue("students", typeof(int));

this.list = (List<Student>)sInfo.GetValue("groups", typeof(List<Student>));

}

public void GetObjectData(SerializationInfo sInfo, StreamingContext contextArg)

{

sInfo.AddValue("id", this.id);

sInfo.AddValue("special", this.special);

sInfo.AddValue("name", this.name);

sInfo.AddValue("students", this.students);

sInfo.AddValue("list", this.list);

}

}

}